WINDCREST®/McGRAW-HILL

MS-DOS® BATCH FILE PROGRAMMING

4th EDITION

DISK INCLUDED

- Over 100 batch files included on disk
- Updated for DOS 6

Ronny Richardson

MS-DOS Batch File Programming

fourth edition

MS-DOS

Batch File Programming

fourth edition

Ronny Richardson

Windcrest®/ McGraw-Hill

New York San Francisco Washington, D.C .Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

005.43
MSR39m
1993

FOURTH EDITION
FIRST PRINTING

©1993 by **Windcrest Books,** an imprint of TAB Books.
TAB Books is a division of McGraw-Hill, Inc.
The name "Windcrest" is a registered trademark of TAB Books.

Printed in the United States of America. All rights reserved. The publisher takes no responsibility for the use of any materials or methods described in this book, nor for the products thereof.

Library of Congress Cataloging-in-Publication Data
Richardson, Ronny.
MS-DOS batch file programming / by Ronny Richardson. — 4th ed.
p. cm.
Includes index.
ISBN 0-8306-4315-X (pbk.) ISBN 0-8306-4588-8 (h)
1. Operating systems (Computers) 2. MS-DOS (Computer file)
3. Electronic data processing—Batch processing. I. Title.
QA76.76.063R54 1993
005.265—dc20 93-179
CIP

Acquisitions editor: Brad Schepp
Editorial team: Robert E. Ostrander, Executive Editor
David M. McCandless, Manuscript Editor
Jodi L. Tyler, Indexer
Production team: Katherine G. Brown, Director
Ollie Harmon, Coding
Tina M. Sourbier, Coding
Rhonda E. Baker, Layout
Lisa M. Mellott, Layout
Lorie L. White, Proofreading
N. Nadine McFarland, Quality Control
Design team: Jaclyn J. Boone, Designer
Brian Allison, Associate Designer
Cover design and illustration: Sandra Blair, Harrisburg, Pa.

WC1

Contents

Acknowledgments xv
Introduction xvii

Part one: Batch file introduction

1 *Batch files are just DOS commands* 3
What are batch files good for? 3
Keystroke reduction 3
New command construction 4
Consistency 4
Safety 6
Summary 7

2 *Creating your own batch files* 8
Picking a name 8
Internal DOS commands 8
Internal CONFIG.SYS commands 9
External commands 10
DOSKEY macros 11
How DOS decides which command to run 11
Where to put batch files 12
Creating batch files 13
The COPY command 13
Edit 14
Using a different editor on batch files 15
Summary 15

3 *Batch file construction* 17
What is a batch file? 17
Batch file formatting 18
Capitalization 18
Spacing 18
Indenting sections 18
Message length 18
Line-length limit 19
Let's write a batch file 19
The problem 19
The solution 19
A second problem 20
The solution 20
Summary 20

Part two: Simple batch file topics

4 *Documentation: "Why in the world did I do that?"* 25
Written documentation 26
Even more documentation 27
Self-documentation 27
Special information at the top 27
Tricky documentation 28
Using labels as remarks 28
Bypassing the comments 29
A warning about these tricks 30
Summary 30

5 *Let's keep it quiet* 31
Command echoing 32
Summary 33

6 *Talking to the user* 34
Displaying messages 34
Telling the user what's going on 35
Keeping the user informed 35
Giving the user instructions 36
Making attractive messages 36
Blank lines 37
What you can & can't echo 38
Pipes 38
Percent sign 38
On/Off 38
Longer fancier messages 38
Summary 39

7 *Using high-ordered ASCII characters* 40
Entering high-ordered ASCII characters 43
What if your editor won't let you? 45
Summary 45

8 *Hold that batch file!* 46
Pausing the batch file 47
Aborting the batch file 47
Summary 47

9 *Applications & problems* 48
Problem 1 48
Problem 2 48
Problem 3 49
Answers 49

Part three: Intermediate batch file topics

10 *Replaceable parameters* 53
- Sending replaceable parameters to a batch file 53
- Using replaceable parameters in your batch files 56
- Limits on replaceable parameters 57
- Uses for replaceable parameters 58
- Replaceable parameters as commands 58
- A couple of examples 59
- Summary 60

11 *Using more than ten replaceable parameters* 62
- The SHIFT command 62
- Batch file tables 63
- What SHIFT can do 65
- Using the SHIFT command 66
- Summary 67

12 *Looping and program flow* 68
- Pseudocode 68
- Naming program lines 69
- The GOTO command 70
  - Looping 70
  - Transferring control 72
  - Quick exit 72
  - Handy formatting option 73
  - Code separation 73
  - Visual separation 73
- The FOR command 74
  - Looping with the FOR command 74
  - Local variable 75
  - Running commands with a FOR loop 75
  - Watching for wildcards in replaceable parameters 76
  - FOR command quirks 77
- Summary 78

13 *The environment* 79
- The prompt 79
- The path 80
- COMSPEC 81
- Environmental variables 81
  - Some notes on names 82

Increasing the size of the environment 82
Deciding on a size 84
Environmental variables are transient 84
Putting variables into the environment 85
Using the environment in batch files 85
What you can do with environmental variables 86
Using the environment when shelling out from a program 86
Environment problem when shelling out 87
Environment benefits when shelling out 88
Environment considerations when shelling out from Windows 89
Using the environment in a batch file 89
Summary 89

14 *Errorlevel* 91
Summary 92

15 *DOS punctuation* 95
Piping & redirection 95
The available piping symbols 96
Devices to pipe to/from 96
How DOS handles a pipe 97
Piping & redirection must use DOS 98
Filters to use in pipes 98
Problems with redirection and piping 99
+ 99
: 100
. 100
@ 100
% 100
%% 100
%Variable% 100
= 100
== 100
Summary 100

16 *Logic testing* 102
Are two values the same? 103
What is the errorlevel? 105
Does a file exist? 107
Does a subdirectory exist? 107
Earlier batch file revisited 108
A.BAT 108
RONNYMD.BAT 109
TOA.BAT 109
Preview of Chapter 18 110
Summary 110

17 *Applications & problems* 112
Problem 4 112
Problem 5 112
Problem 6 113
Answers 113

Part four: Advanced batch file topics

18 *Advanced logic testing* 117
NOT 117
Inter-batch communications 118
Dealing with capitalization 118
GOTO %1 119
Environmental variables 120
The path 120
Run a program 121
Capitalization conclusion 121
Multiple IF-tests 122
A shorter way to show the errorlevel 122
A very short way to show the errorlevel 123
Setting the errorlevel 126
Help 127
Windows 128
Summary 129

19 *Advanced getting information from users* 131
Choices in DOS 6 132
Batcmd: Batch commands 132
Single-character response questions 134
Multiple-character response questions 136
Summary 141

20 *Advanced screen design 1* 143
Text files 143
Writing a program 144
Using a screen compiler 144
ANSI 145
Summary 145

21 *Advanced screen design 2: ANSI.SYS* 147
Introduction 147
Getting started with Ansi 147
Entering Ansi commands 150
Word processors, editors, & "escaping" problems 150
Sending Ansi commands by typing a file 151

Sending Ansi commands using the environment 151
Sending Ansi commands via the prompt 151
Screen control 154
Cursor control 156
Video control 157
Keyboard control 157
An example 160
Summary 163

22 *Batch file subroutines* 164
What's a subroutine, & how do I write one? 165
As a separate batch file 165
Prior to DOS 3.3 165
DOS 3.3 & later 166
An example 166
Inside the calling batch file 171
Storing subroutine commands within a batch file 171
Summary 172

23 *When batch files don't work* 175
Possible error messages 175
Check the spelling 178
Look closer 178
Isolate the problem 179
Split out the problem section 179
Diagram the logic 179
Put it aside for a while 179
Start over 179
Summary 179

24 *Applications & problems* 182
Problem 7 182
Problem 8 182
Problem 9 183
Answers 183

Part five: Improving batch files with software

25 *Batcmd: New batch file commands* 187
Summary 191

26 *Other utilities* 200
ADD 200
ASCII.EXE 200
BS 200
CAPITAL 200
ISITWIN 201

ISITZERO 201
MULTIPLY 201
NEEDHELP 201

27 Builder & Builder Lite 202
Multi-purpose batch languages 203
Builder 203
Bundling 203
Power & flexibility of a programming language 203
Speed of a compiled language 204
Full compatibility with the MS-DOS batch subcommands 204
Ease of learning of the batch language 204
Builder Lite 205
Summary 206

Part six: Batch file applications

28 An automated batch file help system 209
Help files 209
Customized help 212
Context-sensitive help 214
Summary 216

29 Simple batch file menus 217
Structuring your hard disk 217
Introduction to menus 220
Displaying the menu 221
Types of menus 221
Non-resident non-nested menus 221
Resident non-nested menus 223
Hybrid batch file 224
Stand-alone batch file 226
Nested menus 229
Summary 230

30 A document archival system 232
Periodic backups 233
Which file to back up? 234
Types of backup 234
Systematically copy critical data files 236
Never delete old files! 237
Index document files 239
Summary 240

31 Simulating Doskey using batch files 242
Command line macros 242
Command line recorder 246

Running multiple commands 249
Summary 251

32 *Smaller batch file applications* 252
Maintaining your C:\BAT subdirectory 252
Screen saver 252
Is Windows running? 253
Changing subdirectories 253
Running inflexible programs 257
Finding files 259
Running commands occassionally 260
Using the volume in a batch file 264
Batch file floppy-disk catalog 267
Adding to & deleting from your path 269
Password protection 273
Choosing password batch file codes 277
Summary 278

33 *Applications & problems* 280
Problem 10 280
Problem 11 280
Problem 12 281
Problem 13 281
Answers 281

Part seven: Advanced batch file topics

34 *Writing your own batch file utilities* 285
What is a batch file utility? 285
What it needs 285
QuickBASIC 287
Adding 1 to command line number 287
Multiplying two numbers together 290
Builder & Builder Lite 292
Adding 1 to command line number 292
Multiplying two numbers together 294
Turbo Pascal 296
The EnvLib Library Unit 296
Adding 1 to command line number 297
Adding 1 & letting the user pick the environmental variable 299
Multiplying two numbers together 300
Windows 3.1 302
Summary 303

35 *Modifying DOS with DOSKEY & batch files* 306
DOSKEY basics 307
What can I do with DOSKEY? 307
Shortening commands 307
Name changing 308
Function change 308
Putting it together 308
Command line recall 315
I've had it! 316
Summary 316

Part eight: Configuration

36 *CONFIG.SYS file under DOS 3.3* 319
Getting started 319
Problems booting 320
Components of DOS 321
The CONFIG.SYS file 321
Summary 328

37 *CONFIG.SYS file under DOS 5* 329
Summary 331

38 *CONFIG.SYS file under DOS 6* 332
Show us your numbers 332
To load or not to load—that is the question 332
How do I boot thee? Let me count the ways 333
Oops! 333
That's commonly done 334
Defining the menu 335
What does that mean? 336
How long to wait 336
Communicating with AUTOEXEC.BAT 337
Other stuff 338
An example 339
Summary 340

39 *AUTOEXEC.BAT, WINSTART.BAT, & shutdown files* 342
Two minor differences 342
System configuration 343
Memory-resident software 346
Communicating with the CONFIG.SYS file under DOS 6 346
AUTOEXEC.BAT arrangements 346

All-in-one approach 346
One-per-task approach 347
Protecting your AUTOEXEC.BAT file from installation programs 349
WINSTART.BAT 350
LEAVING.BAT 350
Summary 350

Appendices

A Solutions to problems 355

B Batch file documentation 372
Contents of \Bat subdirectory 373
\BAT subdirectories 387
\BLD-ADD 387
\BLD-TIME 387
\ENVLIB 387
\PAS-ADD 387
\PAS-PLUS 388
\BLD-TIME 388
\QB-ADD 388
\QB-TIMES 388

C DOS reference 389

Epilogue 389
Index 399

Acknowledgments

The section on Pascal programming in Chapter 34 was written by Namir Shammas, and I want to thank him for his help.

This book includes a library called the EnvLib Library for allowing Turbo Pascal programs to read from and write to the master copy of the environment. It was developed by Sunny Hill Software and is currently sold by TurboPower Software as part of their Turbo Professional package. I want to thank both companies for allowing me to use it with this book.

I want to thank the folks at hyperkinetix. Chris Bascom (the president) agreed to let one of his programmers, Doug Amaral, write Batcmd and BatScreen especially for my batch books. Doug did a wonderful job on those custom programs, and I appreciate all his work. I think you will find BatCmd and BatScreen to both be wonderful programs. Best of all, you get full-use unrestricted copies of both programs when you buy this book.

Personally, I want to thank my wife Cicinda, my son Tevin, and my daughter Dawna for their support and patience. Because I have only a limited pool of time available, some of the time I spent writing this book had to come from the time I would have spent with my family. They were all very understanding and supportive.

Introduction

Your local bookstore has many books intended to introduce you to PC-compatible computers and the DOS operating system. All of these books spend a lot of time explaining DOS. Then they usually spend about one chapter explaining the basics of batch file programs. One chapter is enough to convince you that batch files are powerful tools; however, it isn't enough to teach you how to use batch files effectively, much less how to write really useful batch files.

Who this book is for

This book will help you learn to utilize the power of batch files. In writing this book, I'm assuming you have a general understanding of DOS. You don't have to be a computer whiz, but you should know how to boot the computer, format a disk, and how to erase and rename files. If you have a hard disk, you should know how to make, delete, and change directories. In general, you should feel comfortable performing simple tasks from the command line prompt. If this isn't the case, you should start with a general book first.

How this book is structured

This book covers one topic per chapter. Sometimes, the topic involves one command, and other times it involves several commands. Most chapters include one or more example batch files.

The first time you read this book, you'll get much more from it if you're in front of a computer. As you read about each new topic, you should enter and run the batch file or run the batch file from the disk that comes with this book. This is also an excellent time to experiment; you'll retain the information much longer if you try to modify it to your own needs. As you read about the various batch files, try to think if you could modify them to better suit your needs or perform some similar task that interests you. This is the best way to learn new material.

Special notes

The book has a number of brief portions of text broken from the rest of the text and marked with icons. These items are very important. Even if you are only "skimming" a chapter, you should read these. You will find different types of messages:

Note: This is a general point of interest you should be aware of.

Remember: These are important points that can cause minor problems if you forget them.

Notice: This is important information on how DOS batch files work.

Warning: These are important points that can cause you significant problems or loss of data if you are not careful.

Hardware and software

Writing a computer book is difficult. I never know what type of hardware and software you have; I don't even know what version of the operating system you are using. In writing about batch files, I'm more fortunate than many writers. Most batch files will run on any hardware configuration, and many of the batch files in this book will run on DOS 3.3 or higher. I've indicated where this is not true. So I have much less to worry about in terms of compatibility, and you can be sure that most of the batch files in this book will work on your system.

For most of the examples in this book, I assume that the files supporting DOS external commands like FORMAT and BACKUP are either in the current subdirectory or in the path when an example batch file requires one of these programs. I also assume that the few utilities programs included with this book are either in the current subdirectory or in a subdirectory that's in your path.

Writing the book

I wrote this book using a Northgate Super Micro 386/20 and printed everything on an HP Laserjet III. The word processor I used for everything except the batch files and tables was Microsoft Word 5.5. I edited batch files and other ASCII files with the extremely nice editor that's built into DOS 6. Finally, I created all the tables using Word for Windows. While too slow for general writing, it has the best table-editing engine I've ever seen.

Most of my batch file testing was on an Apple Macintosh IIcx running SoftPC. SoftPC emulates an 8088 computer on the Macintosh. SoftPC treats each simulated 8088 computer as a single document, much like a word processor treats your letters. I was able to set up multiple 8088 computers on my Macintosh and then install different operating systems on each one. I had DR DOS 5 and 6 and MS-DOS 3.3 to 6 all running on the same Macintosh! While each had their own boot drive (C drive), I configured them all to attach to the same D drive. By copying my batch files and utilities to this drive, I was then able to easily switch between operating systems to test the batch files and utilities under all of the more recent versions of DR and MS-DOS. Try this with a PC and you're stuck booting off floppy disks! All of the problems I encountered were confirmed as operating system bugs by Novell for DR DOS and by Microsoft for MS-DOS. Not one problem was due to running these operating systems on the Macintosh!

Conventions

You should keep a few important pieces of information in mind when reading this book:

- The numbered function keys on the keyboard are shown as F1 through F12. Most older keyboards have ten function keys on the left side of the keyboard, labeled from F1 to F10. Newer keyboards have twelve function keys, F1 to F12, generally along the top of the keyboard.
- Enter stands for the Enter or Return key, which can also be represented on the keyboard as Rtrn or a bent arrow. Most other named keys, like Del or Tab, are also referred to by their name. The directional keys are simply up, down, right, and left arrow.
- Information that you type into the computer and pieces of programs broken out from the regular text are in an alternate typeface, `like this`. The names of keys you hit, like F6 or Enter, and explanatory text is in regular type.
- The caret symbol, ^ , generally means that you hold down the Control (or Ctrl) key and then tap the following key. So ^Z means hold down the Ctrl key and press the Z key. This type of key combination is shown as Ctrl-Z or ^Z.
- Any command inside brackets, [ ], is optional. Brackets are generally used when I'm giving the syntax for a command. In the following, for example,

  ```
  DIR [/P]
  ```

  the /P switch is optional. It causes the listing to pause each time the screen is full. Pressing any key restarts the listing.

- Most of the commands in this book are shown in uppercase when shown on a command line. For the most part, the computer doesn't care. Entering `DIR`, `dir`, or `DiR` are all the same to the computer. About the only time it matters is when you're using the equal sign to compare two strings.

Last note

I've tested all the batch files in this book under multiple versions of DOS. TAB Books and I have done everything possible to ensure that the programs and batch files included with this book and on the disk either run under both DOS 3.3 and higher or are labeled as to which version they require. We have also done our best to make sure that every program and batch file does exactly what we claim it will do. Neither TAB Books nor I make any warranty of any type, expressed or implied, regarding the programs, batch files, and documentation included in this book and on the included disk. In addition, we aren't liable for incidental or consequential damages in connection with, or arising from, the performance of these programs and batch files.

Part one

Batch file introduction

Before you begin to learn to write batch files, these first three chapters will help you understand more global issues relating to batch files.

Batch files are just DOS commands Some of your most useful batch files can be created by simply writing down the DOS commands you use to perform a particular task and then storing that command or those commands in a batch file. This chapter explains that concept in more detail. — **Chapter 1**

Creating your own batch files You have three general ways to write batch files: the COPY command, the editor included with DOS or some other editor. This chapter explains the advantages and disadvantages of each of these approaches. — **Chapter 2**

Batch file construction This chapter explains naming conventions and layout considerations you need to understand when you begin to write batch files. — **Chapter 3**

When you finish these three chapters, you will be able to consolidate one or more commands you would normally enter from the command line prompt into a batch file.

1 Batch files are just DOS commands

The basic idea of a batch file is really very ingenious. When you want the computer to perform a given task more than once, why not have the computer store the steps used to perform that task? A batch file is the way the operating system does that. When you tell the computer to execute a batch file, it reads and follows that batch file just like an actor reads and follows a script. In fact, the concept of the computer following a script is an excellent mental image of a batch file for you to use throughout this book.

What are batch files good for?

Think of this section as a batch file "sales pitch" where I try to convince you that batch files are worth using! As we will see, batch files have four major functions:

- keystroke reduction
- new command construction
- consistency
- safety

We'll look at each of these in more detail.

Keystroke reduction

Let's think about an example. When I sat down to begin writing this chapter, I had to enter some commands first:

C:	I had been doing some "housecleaning" on my D drive. My word processing software is on my C drive so the first thing I did was change to that drive.
CD\WORD	My word processor is stored in the CD\WORD subdirectory, so I then changed to that subdirectory.
WORD	You start my word processor with the command WORD, so I issued that command.

Counting the return needed at the end of each line, I had to type sixteen keystrokes to start my word processor. As we will see, a computerized script (in other words, a batch file) can easily reduce this to as few as two keystrokes. So the first advantage of a batch file is keystroke reduction.

☑ Remember: Batch files can save you a lot of keystrokes.

I'll offer an example to prove this and give you your first experience with batch files. The command to get a directory from the A drive is

```
DIR A:*/P
```

We can shorten this command. Without worrying about what you're typing, enter the following at the command line prompt. Just remember that Enter means press the Enter key (which might also be labeled as Rtrn or Return), and F6 means press the F6 function key and not type "F6" from the keyboard:

```
COPY CON A.BAT  Enter
DIR A:/P        F6 Enter
```

You should see the message "1 file(s) copied." Now put a disk in drive A and type

```
A  Enter
```

If everything goes well, you'll now see a directory of the A drive.

You might be thinking to yourself, "So what? I can simply type `DIR A:/P` when I want a directory." The advantage of A.BAT is that you can get a directory with two keystrokes instead of nine. I used a similar batch file to start my word processor with two keystrokes rather than the sixteen described.

New command construction

When I first started using computers, I had a great deal of difficulty remembering the command to check a disk. Instead of trying to force myself to remember CHKDSK (CHecK DiSK), I wrote a batch file called CHECK.BAT, containing the single command CHKDSK. Typing CHECK instead of CHKDSK saves one keystroke, but that isn't the primary purpose of CHECK.BAT: I can remember the word "check" much easier than I can remember the nonword "CHKDSK."

☑ Remember: Batch files can create new commands or change the name of existing commands.

Of course, I could have renamed CHKDSK.COM to CHECK.COM and achieved the same thing, but then no one else would have been able to check a disk on my computer. Writing a batch file allowed me to use the name I wanted without interfering with normal operations.

In this simple example, "new command construction" is a very fancy name for making a command accessible under another name, much like calling a garbage collector a sanitation engineer or a salesman a purchase consultant. However, as we will see later, long series of commands can be connected to form fairly intelligent sets of commands.

Consistency

Now it's time for a simple one-question test. Put away your DOS manual and don't look at the next figure in this book. What's the solution to this problem: We all know how important it is to back up your hard disk. You should be making frequent full backups, say weekly or monthly, and daily incremental backups. So write down the proper syntax of an incremental backup using the DOS program BACKUP.COM.

4 Batch files are just DOS commands

Remember: A batch file can remember complex syntax for you. ☑

Most of you probably didn't do well on this test. Computer software often uses complex commands on the command line that are difficult for the user to remember. Happily, you don't have to. Figure out the command syntax once and then put it into a batch file and forget it. That way, you'll never have to remember.

By the way, in case you still haven't figured it out, the syntax to make an incremental backup with DOS is

```
BACKUP C:\*.* A: /S /M
```

but you could easily store this in a batch file called I.BAT and just have to remember "I"—which is short for "incremental." If you can't remember that, you can pick another name. With batch files, you get to decide.

I can see you mentally waving a flag. You're asking "Hey, isn't this just another example of new command construction?" The answer is both yes and no. True, we have created a new command called "I" that is easier to remember than the string of switches previously listed, just as Check is easier to remember than CHKDSK. That statement can be made about almost every batch file we will construct in this book. However, the real purpose of this batch file was to always perform the backup consistently—without having to remember the syntax.

Most of the batch files we've looked at so far have one thing in common—they each do only one thing. They don't need to be limited to one activity. DOS keeps track of its location in a batch file even while it's doing something else. Let's do away with that one-command constraint right now. Consider this simple two-line batch file:

```
CHKDSK
DIR
```

First, it runs CHKDSK. While CHKDSK is running, DOS maintains its place in the batch file. Think of yourself putting your finger in a book so that you won't lose your place while answering the phone. After your phone conversation is over, you can pick up reading just where you left off.

When CHKDSK finishes, DOS reads the next command in the batch file and runs DIR. Although CHKDSK takes only a few seconds to run, it could have been a word-processing program that runs indefinitely. It doesn't matter. DOS remembers and picks up the batch file whenever the program finishes.

Remember: Batch files run programs consistently. ☑

In this past example, the batch file established a standard script to be followed each time it is run. While this is not important in this example, consider the next one:

```
C:
CD\WORD
WORD
CD\
MENU
```

This batch file automates loading my word processor, using the steps I have just described. The first three lines are identically the commands I discussed using above when I manually start my word processor. The last two lines reload my menu

program once my word processor has finished. This way, my word processor starts and terminates consistently—always starting on the same drive and in the same subdirectory and always terminating in the root directory of the C drive.

Safety

A final reason for using batch files is safety. Some commands are pretty dangerous. In fact, any operating system is inherently dangerous. Anything that gives you the power to format disks and erase files has the potential for accidents!

Typing `format` instead of `format A:` can destroy the data on your hard disk. Sure, the operating has safeguards and built-in unformatting to help you recover from such an accident, but do you really want to even take the chance of a mishap? Additionally, `erase *.DOC` instead of `erase *.BAK` can erase all your documents and the operating system has no safeguards against this. You'll need the DOS unerasing program or a utility program and some luck to get your files back. If you don't notice the mistake right away, your chances of recovery are greatly reduced. Batch files can form a strong line of defense against the indiscriminate use of these powerful commands.

☑ Remember: Batch files make dangerous commands safer.

You can prevent accidental formatting of a hard disk by first renaming FORMAT.COM to XYZ.COM and then creating FORMAT.BAT with the single command `@XYZ A:`. The @ sign will be examined later, but I'll reveal now that it keeps the XYZ from showing on the screen. If you saw the name, you might be tempted to bypass the batch file and just enter `XYZ` on the command line. We'll find out later how to construct more complex batch files to help prevent erasing our .DOC files.

☑ Remember: Batch files are the easiest programming language to learn!

When most computer users think of programming, they think of spending months to learn languages such as Pascal or C. Like programs created with Pascal or C, batch files are programs. Unlike other programming languages, however, batch file commands are made up of mostly DOS commands, most of which are already familiar to most of you. Therefore, you already know most of the commands you need to write a batch program. Batch files have only a handful of special commands; the vast majority of batch file commands are the regular DOS commands you use every day. If you know how to use DOS, you know most of what you need to know in order to write effective batch files.

When you learn to program in most computer languages, you must learn a great deal before you're able to write even the simplest program. This isn't true with batch files, though. As you'll soon see, you can write some very powerful batch files with standard DOS commands and just a few special batch commands.

Although batch files are easy to write, this ease comes at a price: batch files are limited in what they can do. One of your most common thoughts as you study this book will be "If only . . ." You'll see many situations where you could really automate things if only batch files had specific commands. That is the real drawback to batch files: They're easy to learn and easy to write, but they always seem to stop just short of having enough power.

One way to overcome many of the limitations of DOS batch files is with the many batch utilities available in the public domain as shareware and commercial

programs. Many of these are described in my book *MS DOS Batch File Utilities*, also available from Windcrest. This book includes a copy of Batch Commander, a special batch file utility kit written especially for my batch file books by Doug Amaral, chief programmer at hyperkinetix. Additionally, several small self-written utility programs are also included.

Summary

In summary, batch files have four major functions:

- They can reduce the number of keystrokes needed to execute a command.
- They allow you to combine one or more existing commands into your own specialized commands.
- They allow you to store the proper syntax of command sequence in a script file so you do not have to remember this information.
- They can protect you from dangerous DOS commands.

Batch files are the easiest programs to write because they consist of mostly DOS commands with just a few special batch file commands thrown in.

Creating your own batch files

As we will see, creating batch files is easy. You can create simple batch files using nothing more than the COPY command. For more complex batch files, DOS has a very nice built-in editor that does a fantastic job. For even more complex batch files (and there are very few this complex), chances are the word processor you currently use will edit batch files as well. However, before we get into the mechanics of creating the batch files, we need to explore a couple of other issues.

Picking a name

The easy part of picking a name is picking an extension. DOS gives you no choice except for .BAT. When picking a name, you want to choose one that describes the function of the batch file as well as possible, given the constraints of the eight-character limitation on filenames. If you're naming your word processor documents, a specific extension and eight characters for the name is probability the only limitation you face. However, batch file authors face other limitations.

DOS will accept four kinds of commands: internal commands, internal CONFIG.SYS commands, .EXE and .COM external programs, and batch files. Beginning with DOS 5, DOSKEY macros are added to this list. In choosing names for our batch files, we have to avoid conflicts with the other four categories of commands. Let's look at each of these four categories individually.

Internal DOS commands

Internal commands are commands so important that they're built directly into DOS. Every time DOS receives a command, it first checks to see if that command is internal, like ERASE. If so, then it executes that command.

Internal commands represent a trade-off. If DOS had every possible command built-in, then it would take up so much room that there would be no room left for application programs. If no programs were built into DOS, however, every system

disk would require numerous programs to perform every simple task. This would waste a lot of space. The trade-off made by DOS is that the most important commands are built-in, while the remaining commands are external programs that require a .EXE or .COM file to run.

The names for internal commands are part of DOS, which is why you can't name your batch file the same name as an internal command. If you do, there's no way to execute your program. If you created a file named COPY.BAT and tried to run it with the command:

```
COPY
```

then DOS would automatically go to the built-in DOS COPY command, and try to copy nothing to nothing. If you tried to run it using

```
COPY.BAT
```

then DOS would try to copy a file named .BAT to nothing and thus would return an error message.

Remember: If you give your batch file the same name as a DOS internal command, you will be unable to run your batch file. ☑

TABLE 2-1 lists the commands that are so important that DOS makes them internal commands. Remember that these words cannot be used as the name of a batch file.

Table 2-1
DOS internal commands.

BREAK	DISCOMP	MICRO	SET
CALL	ECHO	MKDIR	SHIFT
CD	ERASE	NET	TIME
CHCP	EXIT	PATH	TRUENAME[1]
CHDIR	FOR	PAUSE	TYPE
CLS	GOTO	PROMPT	VER
COPY	IF	RD	VERIFY
CTTY	LH	REM	VOL
DATE	LOADHIGH	REN	
DEL	MAIL	RENAME	
DIR	MD	RMDIR	

[1]This is an undocumented command that will display the true name and path to a file while ignoring any drive assignments.

Internal CONFIG.SYS commands

Internal CONFIG.SYS commands are also commands that are built into DOS, just like internal commands. However, these commands can only be accessed by the CONFIG.SYS file. That means most of their names can be used for batch file names without conflict. Three of them (BREAK, REM, and SET) duplicate internal commands and so cannot be used for batch file names. TABLE 2-2 lists the internal CONFIG.SYS commands.

Table 2-2
DOS internal CONFIG.SYS commands.

BREAK	DOS	LASTDRIVE	SET
BUFFERS	DRIVPARM	MENUCOLOR	SHELL
COMMENT[1]	FCBS	MENUDEFAULT	STACKS
COUNTRY	FILES	MENUITEM	SUBMENU
DEVICE	INCLUDE	NUMLOCK	SWITCHES
DEVICEHIGH	INSTALL	REM	

[1]This is an undocumented command. The command "COMMENT=XY" causes the CONFIG.SYS file to ignore any command line beginning with "XY." Note that only the first two characters are significant and it is case insensitive.

External commands

If a DOS command is not an internal command built into the operating system itself, then it is an external command that exists as a .COM or .EXE program in your DOS subdirectory. You don't want a situation where the same name is used for a .COM and .BAT file or an .EXE and a .BAT file. TABLE 2-3 lists the DOS external commands.

Table 2-3
DOS external commands.

APPEND	EDLIN[1]	MAIL	PRINT
ASSIGN	EMM386[2]	MEM	QBASIC
ATTRIB	EXE2BIN[1]	MEMMAKER	RECOVER[1]
BACKUP	EXPAND	MICRO	REPLACE
CHKDSK	FASTOPEN	MIRROR	RESTORE
CHOICE	FC	MODE	SETVER[2]
COMMAND	FDISK	MORE	SHARE
COMP[1]	FIND	MSAV	SIZER[2,3]
DBLSPACE	FORMAT	MSBACKUP	SMARTDRV
DEBUG	GRAFTABL	MSD	SMARTMON[4]
DEFRAG	GRAPHICS	MSHERC	SORT
DELOLDOS	HELP	MWAV	SUBST
DISKCOMP	INTERLINK[2]	MWAVSTR	SYS
DISKCOPY	INTERSVR	MWBACKUP	TREE
DOSHELP	JOIN[1]	MWUNDEL	UNDELETE
DOSKEY	KEYB	NET	UNFORMAT
DOSSHELL	LABEL	NLSFUNC	VSAFE
EDIT	LOADFIX	POWER[2]	XCOPY

[1]These are no longer part of the main portion of DOS; however, they are supported on the supplementary disk.

[2]This command is both a device driver and an external utility.

[3]SIZER is part of the MEMMAKER utility.

[4]This is a Windows program that is included with DOS.

While you don't want to use the same name for a batch file that is already in use for a program, external commands are not nearly as intractable as internal commands. With an internal command, the only way to change the command is to modify the contents of the DOS operating system files—a topic too advanced for this book.

However, it is easy to alter external commands. Just rename FORMAT.COM to XYZ.COM, and the name of the command changes. Now, create a batch file called FORMAT.BAT and you will have no conflict. As you'll see later on in this book, you sometimes have very good reasons for doing this.

So, my point with external commands is to simply avoid any name conflicts. You'll want to avoid the same conflicts with all your other software as well. If the file that runs your word processor is WP.EXE, then you'll want to avoid creating WP.BAT because you'll end up running WP.EXE under some conditions and WP.BAT under others. While it's been said that consistency is the hobgoblin of small minds, it seems reasonable to me that you would always want the WP command to do the same thing on your system.

DOSKEY macros

DOSKEY is a memory resident program that was added to DOS in version 5. DOSKEY gives you the ability to write macros to shorten commands. For example, the command:

```
DOSKEY A=CD\LOTUS\PROJECTS\MILTON
```

will cause DOS to issue a `CD\LOTUS\PROJECTS\MILTON` each time a command line starts with an "a" or "A." It is interesting to note that DOSKEY macros supersede and can therefore "turn off" internal commands. If you issue the command

```
DOSKEY DIR=ECHO DIR Command Turned Off
```

then the message "DIR Command Turned Off" is displayed each time the user requests a directory.

As an aside, it is interesting to note that DOSKEY is more particular about how its commands are entered than DOS is. While DOS treats "DIR" and " DIR" and "DIR/P" and "DIR /P" as the same commands, DOSKEY does not. As a result, if you have "turned off" the DIR command with the above DOSKEY macro, you can still get a directory with the commands " DIR" or "DIR/P." If you have a batch file with the same name as a DOSKEY macro, then either of these tricks will allow you to run the batch file without having to alter its name or the name of the DOSKEY macro.

How DOS decides which command to run

Because the issue has come up, let's take some time and look at the process DOS goes through in deciding what to run when you give it a command. When you issue a command, DOS first checks to see if that command is a DOSKEY macro. Next, it checks to see if it is an internal command. If it finds the command as an internal command, then it runs it.

This explains why COPY.BAT (just mentioned) never runs: DOS never checks the drive because it finds Copy in its internal list of commands.

If the command isn't an internal command, DOS next checks the current subdirectory for a .COM file by that name, then an .EXE file, and finally a .BAT file. If DOS finds a program with the correct name, it executes that program. If DOS doesn't find a file with the correct name in the current directory, then it searches the PATH subdirectory by subdirectory, looking for a .COM, .EXE, or .BAT file (in

that order) in each subdirectory. If DOS finds a program in the PATH with the correct name, it executes that program. Otherwise, it returns the "Bad command or file name" error message. TABLE 2-4 illustrates this.

Table 2-4
Hierarchy of DOS commands.

1	A DOSKEY macro
2	An internal command
3	A *.COM file in the current subdirectory
4	A *.EXE file in the current subdirectory
5	A *.BAT file in the current subdirectory
6	A *.COM file in the C:\FIRST subdirectory
7	A *.EXE file in the C:\FIRST subdirectoy
8	A *.BAT file in the C:\FIRST subdirectory
9	A *.COM file in the C:\SECOND subdirectory
10	A *.EXE file in the C:\SECOND subdirectory
11	A *.BAT file in the C:\SECOND subdirectory
12	A *.COM file in the C:\THIRD subdirectory
13	A *.EXE file in the C:\THIRD subdirectory
14	A *.BAT file in the C:\Third subdirectory

Let's look back at our earlier example of WP.EXE and WP.BAT. Assume that WP.EXE is in the C:\WP subdirectory and that WP.BAT is in the C:\BAT subdirectory. If the path is C:\;C;\DOS;C:\BAT;C:\WP, then WP.BAT will usually be run in response to the WP command because its subdirectory comes earlier in the path. However, if you look at lines 2-4 in TABLE 2-3, when the user is in the C:\WP subdirectory, then WP.EXE is run instead of WP.BAT in response to the WP command.

This uncertainty about whether WP.EXE or WP.BAT will run is why you should avoid naming conflicts. In this example, if the purpose of WP.BAT is to run WP.EXE in a particular fashion, then WP.EXE could be renamed and the new name built into WP.BAT to avoid this conflict.

☑ Remember: If you give your batch file the same name as a DOS external command or any other .EXE or .COM program, the batch file will conflict with that program. When that happens, the batch file or the program might run in response to the command—depending on which subdirectory you are in.

Where to put batch files

Batch files are designed to help you automate everyday processes, which means that you must be able to run your batch files at any time in any location. Thus, they must be in a subdirectory in your path. You could put them in your DOS subdirectory, but the batch files might get erased when you upgrade the operating system. Besides, that subdirectory already has a lot of files from just DOS, so why make things even more crowded?

I recommend that you give your batch files their own subdirectory called \BAT. Regardless of what you end up naming it, you're going to want that subdirectory to be listed near the front of your path.

Remember: Batch files belong in the C:\BAT subdirectory, and that subdirectory belongs in your path as one of the first listings. ☑

As a general rule, you want the subdirectories in your path listed in order of frequency of use. Subdirectories with commands you use a lot should be listed first, while subdirectories with less frequently used commands can be listed last. Many users will want their DOS subdirectory first and their batch subdirectory second.

Creating batch files

You can create batch files through three main ways:

- ❑ Use the COPY command
- ❑ Use Edit, the editor that comes with DOS
- ❑ Use your own word processor

Each of these approaches has advantages and disadvantages. We'll look at each one in turn.

The COPY command

One of the most basic DOS commands is the COPY command. You've probably used this command to copy files in the past. For example,

```
COPY A:*.* B:
```

copies between two DOS logical devices, both of which happen to be files. The COPY command can be used to copy between any DOS logical devices. In addition to files, there are other logical devices:

CON	Depending on the usage, this is either the screen or the keyboard. When used as the target, it's the screen. (After all, you can't copy to the keyboard!) When used as the source, CON is the keyboard.
PRN	The PRN specification refers to your printer.
NUL	This is DOS's version of a black hole. Anything that's copied to NUL is gone. I'll show you later how to send unwanted messages to NUL.

You can copy between files and any of these DOS logical devices or between the logical devices. (Not all combinations make sense—for example, copying from NUL or PRN.) You can view your CONFIG.SYS file on the screen by typing the following:

```
COPY C:\CONFIG.SYS CON
```

This command copies the file to the screen. You can turn your printer into a simple typewriter with the command:

```
COPY CON PRN
```

When you're finished typing, press the F6 function key—followed by Enter—and what you've typed will be sent to the printer. All the combinations that work with COPY, however, don't work with XCOPY. Unlike COPY, XCOPY can only copy between files.

Remember: XCOPY cannot be used to create batch files from the console the way that COPY can. ☑

You can create simple batch files by copying from the keyboard (CON) to a file. To create a simple batch file called EXAMPLE1.BAT, enter the following command:

```
COPY CON EXAMPLE1.BAT
```

What you're telling DOS with this command is copy from the console to a file called EXAMPLE1.BAT. Note that the command line prompt doesn't return when you press Enter. Now enter

```
DIR *.*
```

then press the F6 function key and Enter. You should see the message "1 File(s) copied." That's it: you've created the file. If you issue a DIR command, you'll see EXAMPLE1.BAT listed among your other files.

☑ Remember: You can use the COPY command to create short batch files or other short ASCII files.

You can use this method to create batch files of any length. In fact, you can use this method to write a file containing a letter or anything for which you'd use a word processor. You could, but you probably wouldn't want to. Once you press Enter, there's no way to go back and edit that line. And once you create a file, there's no way to edit it. If you make a mistake, you must start over! In spite of the fact that editing is limited to the current line, though, COPY CON is an effective way to create short batch files.

Edit

When you need to create a longer batch file, or if you need to edit an existing batch file, COPY CON isn't good enough. You need an editor. Prior to DOS 5.0, DOS came with a simple line editor called Edlin. "Line editor" means it lets you edit only one line at a time. Most editors and word processors are full-screen editors. They let you edit anywhere on the screen. For this reason, most people don't like Edlin.

With the release of DOS 5.0, Microsoft added a new editor, Edit. Unlike Edlin, Edit is a full-screen editor with a very Windows-like appearance. Users of Microsoft QBasic will be immediately familiar with Edit, because it works like just like QBasic. In fact, EDIT.COM just performs some minor configuration and then loads QBasic to perform the actual editing.

Edit supports a mouse. There's even a "mouse elevator" on the right side of the screen for rapid movement around files. No matter what you're doing in Edit, context-sensitive help is available simply by pressing F1.

Edit has four menus: File, Edit, Search, and Options. The File menu has the normal options to load and save files. The Edit menu has the following four options:

Cut	This cuts the highlighted text out of the document into the scrap.
Copy	This copies the highlighted text to the scrap without disturbing the highlighted text.
Paste	This copies the text in the scrap to the document at the cursor position. It doesn't disturb the contents of the scrap, so you can make multiple copies.
Clear	This clears the contents of the scrap.

The Search menu has three options:

Find	This lets you locate the next occurrence of text.
Repeat Last Find	This lets you find the next occurrence of text after searching once.
Change	This lets you perform a search and replace on text.

The Options menu lets you change the path to the help file and change the colors displayed on the screen. Edit allows you to edit files as large as will fit into memory. I've successfully used it to edit files as large as 250K without any problems. All in all, it's a fantastic replacement for Edlin.

Using a different editor on batch files

Occasionally, you are going to want to use your word processor to edit your batch files instead of Edit. Edit does such a nice job that those occasions will be rare. However, I usually run my finished batch files through the spell checker in my word processor. In addition, I load batch files into my word processor when I want to insert boxes because my word processor has a box-drawing command that can insert the high-ordered ASCII characters very easily.

Most full-featured word processors, like Wordstar or Microsoft Word, can be used to edit batch files. Normally, Wordstar doesn't produce ASCII files. If you copy a Wordstar file to the screen, you'll see that the last letter of most words looks funny. To produce ASCII text with Wordstar, you must open your document in what Wordstar calls the nondocument mode. This means that you can't open your batch file at the same time you start Wordstar, using the short-cut:

```
WS BATCH.BAT
```

Wordstar lets you specify the filename when you start the program and thus avoid the opening menu. But this method automatically puts you in document mode, and you won't get an ASCII file. Even in nondocument mode, Wordstar won't produce an ASCII file if you use the paragraph reformat command, ^B. However, if you follow these few precautions, Wordstar will work well as a batch file editor.

With Microsoft Word, you can easily produce ASCII files as well. If you load an existing ASCII file into Word, it automatically treats it as straight ASCII. To create an ASCII file new within Word, just save the file using the "Save as" option and select the "Text only" option. Word has difficulty with ASCII files with long lines, however. While it saves them properly without the wordwrap, it cannot display them without wordwrap. That can make looking at the lines on the screen very confusing.

Most other word processors will produce ASCII files. However, ASCII probably won't be the default mode. It might be a separate mode, which will usually have wordwrap turned off, or you might need to translate your document into ASCII format. A simple word processor might work similar to a typewriter with limited editing ability. If you're in doubt, create a small document and then copy it to the screen. If you see exactly what you typed, then it's ASCII.

Batch files should not have the same name as an internal command.

Summary

It's generally not a good idea to give batch files the same name as an existing .COM or .EXE program.

If a batch file has the same name as a DOSKEY macro, you can still run it by placing a space before its name on the command line.

When deciding which programs to run, DOS first looks for DOSKEY macros and next for internal commands. If the command is not one of these, then DOS looks for a .COM program, a .EXE program, or a batch file in the current subdirectory, in that order. If the command is not a DOSKEY macro, an internal command, a program, or a batch file in the current subdirectory, then DOS looks for a .COM program, a .EXE program, or a batch file in each subdirectory on the path, in the order listed in the path.

Batch files are best located in their own subdirectory, generally called \BAT. This subdirectory should be in the path.

Simple batch files can be created quickly with COPY.

More complex batch files can be created with the DOS Edit program.

In addition, most word processors have the ability to edit batch files, although you generally have to use a special unformatted or programming mode.

3 Batch file construction

So far, you have been introduced to some good reasons to use batch files and you've seen how to use the nice editor that's included with DOS. Now, it's time to begin writing your own batch files. But before that, we'll spend a few minutes talking about how to structure them.

What is a batch file?

A batch file script is a special file containing one or more DOS commands and a few batch subcommands, with each command on a separate line. That's it—nothing more and nothing less. This is illustrated with the following:

```
First command
Second command
Third command
Fourth command
Fifth command
(and so on)
```

You run a batch file by entering its name at the command line prompt. You can enter the .BAT extension, but you don't have to. For example, to run a batch file named FRED.BAT, you would enter

```
FRED
```

Some batch files require additional information, which you must enter after the batch file name. This additional information is called *replaceable parameters*. The batch file name and each separate piece of information must be separated by either a space or a comma; for example,

```
FRED Yes,1,*.BAK No
```

☑ Remember: Replaceable parameters are data you enter on the command line after the name of the batch file.

Batch file formatting

DOS doesn't require batch files to have any special form of formatting. The computer doesn't care either: it will successfully execute any properly written batch file, no matter how it is formatted. The only reason to format batch files in any special way is for human readability.

What follows is the scheme I've developed for use with my batch files. It makes the batch files easily readable, while using the limited formatting that DOS allows. If you develop a different system that works better for you, feel free to use it.

Capitalization

I use ALL CAPITAL LETTERS in both the batch files and the messages they display to draw attention to particularly important elements. In general, I use capitalization in the following fashion:

- ❑ I use all capital letters for DOS commands and program names in the batch files. They are also capitalized in this book.
- ❑ I capitalize the first letter of each word in most messages and remarks.
- ❑ When a message tells you what to enter on the command line, I capitalize the command line.
- ❑ When a message is very important, I capitalize the entire message; for example, "WARNING: THIS TAKES AN HOUR."

Spacing

In a long program, it can be difficult to see the different sections if you enter text on each line. To visually break the program up into different sections, I leave one or more blank lines between each section. These blank lines don't affect the operation of the batch file because DOS ignores them, but they do make it much easier to read.

While the batch files in this book do have blank lines between the sections, I've left those blank lines out of the table listing I use for the longer batch files. I've found that, while blank lines enhance the readablility of programs, they don't enhance the tables.

Indenting sections

I've also found that it can be useful to indent the lines of a section between the beginning and ending label of the section. If you do that, a batch file segment might looks like this:

```
:HELP
 ECHO This Runs Your Backup Program
 ECHO You Must Start It With Either An F or I
 ECHO On The Command Line
 ECHO The F Is For A Full Backup
 ECHO The I Is For An Incremental Backup
GOTO END
```

As you can see, this clarifies which statements belong in this section, something that's especially useful in longer programs.

Message length

You will notice that most of the messages in my programs have fairly short lines. For longer messages, I use multiple lines. It's been my experience that shorter lines are easier to read on the screen than longer ones. It's a lot like printing your documents in multicolumn format, except the batch files only use the left column.

Line-length limit

Commands entered from the command line are limited to 127 characters. (Technically, the limit is 128 characters, but the last character must be the return you must press to send the command to DOS.) Batch files are limited to this same limitation: No batch file command can exceed 127 characters. As we will see later in this book, DOS treats some lines in your batch file as being longer than their physical length. That can sometimes make it difficult to gauge the actual length of the line, but the line must still be 127 characters or less after DOS expands it to its final length.

If a line exceeds this limit, one of two things are likely to happen. Either the computer will lock up, or DOS will truncate the command after the 127th character. In either case, the batch file will not perform as you expected.

Fortunately, you'll rarely encounter this problem. However, it's a good idea to keep this limit in the back of your mind in case you ever find yourself typing in a very long line.

Let's write a batch file

So far in this book, I've have not introduced a single batch file command. I'm not going to introduce any until the next chapter. Yet, you can still write some useful batch files. You have already seen a couple of useful batch files: I.BAT, which performs an incremental backup, and CHECK.BAT, which runs the DOS Chkdsk program. Both of these are included on the disk that comes with this book. Now, let's look at two more situations where batch files are useful.

The problem

A number of major software packages like Lotus, Word, WordPerfect, or WordStar let you run COMMAND.COM to shell out to DOS to perform some minor task. The original program stays in memory so only a small amount of free memory is available. Generally, it's difficult to tell you shelled out of another program so sometimes users are tempted to start another program from this second DOS prompt. However, you either don't have enough memory to load another program or memory is so scarce that the second program runs poorly.

The solution

A few programs take care of the problem for you. Windows 3.1 will not let you load a second copy of Windows; although, it will happily let you run any other program. Additionally, if you create an environmental variable called "WINPMT" then Windows 3.1 will use this as the prompt when you shell out of Windows so a WINPMT value like "Type EXIT To Return To Windows$P$G" will give you a constant reminder that Windows is loaded. A few other programs automatically modify your prompt when you shell out.

However, most programs operate like Lotus does and require you to build in whatever protection you desire. You want to make it plain that you are shelled out of Lotus while running DOS under Lotus but Lotus will not do that by itself. If you start Lotus with a batch file, it's easy to do just that. Take the following steps in the batch file:

- Change to the drive containing Lotus. This will be a command like "C:".
- Change to the subdirectory containing Lotus. This would be a command like `CD\123` or `CD\LOTUS`, depending on how you have your hard disk structured.
- Change the prompt from the usual PG to a reminder that Lotus is running. This would be a command like `PROMPT Type EXIT To Return To Lotus$_$P$G`. You can't see this prompt in Lotus; but if you shell out of Lotus,

it uses the same prompt that was present when Lotus started, so you will see the reminder.

- Start Lotus. This would be a command like `123` to bypass the initial menu or `LOTUS` to go through the initial menu.
- Once Lotus terminates and the batch file has taken back over, reset the prompt. This would be a command like `PROMPT $P$G`.

None of these steps requires a batch file command, so you already know enough to write this batch file. If you have Windows on your system, you might even want to put this book down and fire up Editor and give it a shot. Go ahead; the rest of us will wait.

The resulting batch file requires five lines, one for each of the things we just listed:

```
C:
CD\LOTUS
PROMPT Type EXIT To Return To LOTUS$_$P$G
123
PROMPT $P$G
```

With the "Type EXIT To Return To Lotus C:\>" prompt showing, it will be hard for you to accidentally run a second copy of Lotus. By the way, this solution is not limited to Lotus. This technique can be used with any program you start with a batch file.

A second problem

I like to perform an incremental backup at the end of every day using the XCOPY command. However, there are certain files that get modified every day that I don't need to backup. These are

- MIRROR.FIL. This file is updated by DOS 6 every time I boot by a command in my AUTOEXEC.BAT file. It contains information about the file allocation table that DOS uses to unformat the hard disk and unerase files.
- *.BAK. I'll have a backup of the original file so I don't need a backup of the backup file.
- 386SPART.PAR. This is my Windows permanent swap file and is modified each time I run Windows.

I need a backup batch file that first removes the archive setting for these files and then performs the incremental backup.

The solution

The first step is to make sure the computer is in the root directory of the current drive. After that, it takes three lines to remove these archive settings. The operational segment of PREPARE.BAT, which does just this, is shown next:

```
CD\
ATTRIB -A MIRROR.FIL
ATTRIB -A *.BAK /S
ATTRIB -A 386SPART.PAR
```

If you like, one more line, `XCOPY *.* A: /S /M`, could be added to perform the actual backup from the same batch file.

Once again, this batch file is nothing more than the same commands you would enter at the command line to perform this same task. However, once you've figured out how to do it once, you can record those commands in a batch file and forget

about it because the batch file remembers everything for you. Additionally, entering the name of the batch file takes far fewer keystrokes each afternoon than entering all these commands.

Summary

Batch files can be nothing more than a series of DOS commands identical to the commands you would enter on the command line.

Batch files can be created by writing down the commands you used at the command line prompt and then entering them into a batch file.

Proper formatting can make batch files more readable to humans without affecting the computer's ability to execute them. However, the formatting DOS allows in a batch file is very limited.

Consistent capitalization, blank lines between parts of the batch file, and indenting the commands within a section all contribute to a readable batch file.

Very useful batch files can be written without using a single special batch command.

Part two

Simple batch file topics

Now that you've seen how to name, write, and structure batch files, it's time to look at some of the simpler batch subcommands. A batch subcommand is a DOS command designed to run inside a batch file. However, most of them will also work from the command line if you have a use for that.

REM documentation "Why in the world did I do that" The batch language only has a few commands of its own but it can also run any program you own. As a result, some batch files tend to become very long. Using the REM command to document those longer batch files can make following their logic easier when you have to modify them in the future. This chapter explains how to do that.

Chapter 4

Let's keep it quiet The default mode of operation for a batch file is to have command echoing on, which means that commands are displayed on the screen before they are executed. By turning command echoing off, your screens will look much neater. This chapter explains how to do that.

Chapter 5

Talking to the user Batch files frequently need to "talk" to the user to display information, ask questions or provide warnings. The ECHO command provides the method for doing this. This chapter explains how to use the ECHO command effectively.

Chapter 6

Using high-ordered ASCII characters Your batch files can be made much more attractive by using some of the high-ordered ASCII characters available through DOS. This chapter explains how to use these characters.

Chapter 7

Hold that batch file! Sometimes you need for the batch file to wait for the user to catch up. Maybe it has put a lot of information on the screen for the user to read, or maybe the user needs a chance to press Ctrl-Break and halt the batch file. This chapter explains how to use the PAUSE command in a batch file to suspend its operation until a key is pressed.

Chapter 8

Applications and problems Before moving on to the intermediate batch subcommands, you should understand the commands and concepts covered so far. This chapter is a quick test. Pass it, and you're ready to move on.

Chapter 9

Documentation "Why in the world did I do that?"

One of the worst possible feelings a programmer can have is going back to an old program and not understanding how it works. Basic programmers even have a name for the type of code most likely to cause this condition: "spaghetti code." It's called spaghetti code because it twists, turns, and winds around like cooked spaghetti.

In any programming language, it's important to document your work—both written documentation and documentation inside the program. If you look at the source code for any well written program in any language, you'll see lots of documentation, shown as comments. Comments are lines of text inside the program that have nothing to do with the actual program. They're simply intended to help you understand the surrounding code, especially when you haven't seen the program for six months.

Internal documentation for a batch file consists of the remark line, started with the REM command. If you start a line with REM followed by a space, you can enter almost anything you want to. The only thing you can't use are the redirection (piping) characters. The computer will skip the remark line, so the information on that line are simply comments for anyone who has to go back to modify the code.

Take a look at the following batch file:

```
:START
 ERASE %1.BAK
 COPY %1.DOC B:
 COPY C:\WORD\NORMAL\%1.STY B:
 ERASE C:\WORD\NORMAL\%1.BAK
 COPY C:\WORD\%1.CMP B:
```

```
 SHIFT
 IF (%1)==() GOTO END
 GOTO START
:END
```

This example includes some commands you haven't learned yet, but they're not the important part. The important thing is that %1 takes the place of a filename while the batch file is running. Given that, do you know what the file is doing? Probably not. Now consider the same batch file, properly documented, next:

```
REM YESDOC.BAT
REM Batch File To Copy New Files To Floppy Disk
REM :START Is Top Of Loop
:START
 REM Erase The .BAK Backup File
 ERASE %1.BAK
 REM Copy The Document File To Floppy
 COPY %1.DOC B:
 REM Copy The Associated Style Sheet To Floppy
 COPY C:\WORD\NORMAL\%1.STY B:
 REM Erase .BAK Backup Of Style Sheet
 ERASE C:\WORD\NORMAL\%1.BAK
 REM Copy Document Dictionary To Floppy
 COPY C:\WORD\%1.CMP B:
 REM Get Next Name
 SHIFT
 REM Test For Another Name, Exit If Not Found
 IF (%1)==() GOTO END
 REM Loop Through Next Name
 GOTO START
:End
```

It's the same batch file; none of the working commands are any different. But now you can probably understand the batch file without my explaining the commands I haven't yet covered.

This isn't a made-up example; the first batch file is an actual working batch file that's used every day. The author of the batch file could probably modify it, but it isn't likely that anyone else could. The point is that batch files aren't so simple that you can skip documentation. No one ever suffered from too much documentation.

☑ Remember: Use the REM command inside your batch files to document the logic of the batch file.

All the remark command does, therefore, is allow you to add comments to your batch file. The syntax is "REM" followed by a space and then your comments. Remember, the line length must be 127 characters or less. In the next chapter, we will see how to keep remark lines from showing when you run your batch files.

Written documentation

You can also use written documentation, one example of this being the tables in this book that list the batch files. The code is on the left, and a detailed explanation is on the right. Of course, you probably won't need to document your batch files in this level of detail but might find it adequate to simply print out a listing and write notes in the margins.

Remember: If your batch files are used by others, written documentation can help them use those batch files more effectively.

A third form of documentation is a printed user's manual. In Appendix B, you'll find a brief description of what each program that comes with this book does and what it needs as an input. If you use a lot of batch files on your system, you might find it useful to keep a similar listing of your own programs. You might also find it useful to keep this sort of documentation on the utility programs you add to your system.

Even more documentation

Over the course of the years, I've written six books for TAB dealing with batch files. While working on all these batch file books, I've written and debugged over 700 batch files! In addition, I've looked at and debugged countless batch files written by others. Because of my diverse experience with batch files, I've developed some ideas on what they should look like.

Self-documention

DOS 5 has added a nifty new feature to most of its commands. If you're not sure what they do or how to use them, you can start them with a /? switch and get a screen of helpful information. This makes it quick and easy for the user to find out what a program does or what inputs it needs. I expect that, over time, this feature will be added to most programs.

Implementing it in a batch file is fairly easy. A section of code near the top of each of the batch file tests to see if the user entered a /?. If it is, the batch file displays a help screen and exits to DOS. I use the /? switch to be consistent with DOS. In Chapter 18, after covering some more basics, we'll see the details of how to add this to batch files.

Special information at the top

Some information is important enough that every batch file must have it. The importance of this information suggests that we put it at the top of each program:

- **Name.** The first line of the program gives the name of the batch file. This isn't extremely critical because you know the name from the DOS filename; however, it is useful information—especially if your editor does not display the name of the file you are editing. In addition, some very useful batch files in the interior of this book will use this information. These batch files will be much more useful if this line is included in each batch file.
- **Purpose.** The second line of the program gives the purpose of the batch file. In this book's sample programs, I've kept the purpose line fairly short and used multiple lines for a longer purpose in order to make the tables look good. For reasons we'll see later in the book, you should use as descriptive a purpose statement as possible but still keep the entire line below 80 characters long. This is not absolutely necessary, but some batch files shown later in the book use this information and work better with line lengths below 80. It's ok to use multiple lines for the purpose as long as each line begins with a "REM PURPOSE:".
- **Version.** The first version of a program is 1.00. A tiny change would make it 1.01, a larger change would make it 1.10, and a major change would make it 2.00. Because programs tend to evolve over time, this line is a good indicator of how long a batch file has been in use.

  You might notice that many of the batch files in this book are version 1.0. This isn't because I write perfect programs each time but rather because my programs undergo extensive beta testing before they are added to my book. As is common with many software vendors, I begin numbering my beta copies at

0.00 and only mark them as 1.00 after they have finished this extensive beta testing. As a result, they don't get a number higher than 1.00 unless I enhance them, fix a bug after beta testing, or go back and figure out a better way to do something.

- **Date.** This is the date of the last modification to the program. A program that has not been updated for a very long time is either very stable or not used very often.
- **Revision.** Recently, I've started adding a fifth line of documentation below the date. I call that line update "revision." When I make a change to a batch file, I now not only update the version and date, I add a note about the reason for the update. That makes it easier to follow the evolution of my batch files over time. This is a recent addition so not all the batch files on the disk have this line (none of the version 1.00 would because they have not been revised) and those that do may not have notes back to 1.00 because changes may have been made before I started doing this.

Following this modified scheme, the top of a typical batch file will look something like this:

```
REM NAME:     I.BAT
REM PURPOSE:  Perform An Incremental Backup
REM VERSION:  1.04
REM DATE:     November 25, 1992
REM REVISION: 1.01: Added *.OBJ To Ignored Files
REM REVISION: 1.02: Added *.INI To Ignored Files
REM REVISION: 1.03: Added Processing Messages
```

☑ Remember: Adding the name, purpose, version number, date created/modified and revision data to the top of each batch file is an excellent form of documentation. If you study the sample programs that come with this book, you will see that almost all of them follow this documentation scheme.

Tricky documentation

While I don't recommend it, you can use two tricky ways to add documentation to your batch files—both of which will speed up your batch file execution slightly. Because the operating system processes a batch file line-by-line, both tricks involve working around this line-by-line processing of the REM commands:

Using labels as remarks

When a line begins with a colon, the operating system knows that line is a label; thus, it only needs to process labels when processing a GOTO command. When the batch file is not processing a GOTO command, it skips any line that begins with a colon. (GOTO is examined in detail in Chapter 12.) One way to do this is just to start each documentation line with a colon, like this:

```
:Name:    BATCH.BAT
:PURPOSE: Show Using Colon To Mark Documentation
:VERSION: 1.00
:DATE:    May 28, 1992
:None Of These Lines Will Execute Or Show On The Screen
:So They Function Similarly To Rem Commands
```

However, this can cause problems later on if you plan on using a label like :NONE that conflicts with one of your documentation lines. The colon is also hard to see in printouts, so you might not notice these as comments right away. A solution is to combine the colon with a character that is invalid in a label, like an asterisk. That

way, it's more visible and has no chance of conflicting with a label. The resulting batch file segment would look like this:

```
:* Name:     BATCH.BAT
:* PURPOSE: Show Using Colon To Mark Documentation
:* VERSION: 1.00
:* DATE:     May 28, 1992
:* None Of These Lines Will Execute Or Show On The
:* Screen So They Function Similarly To Rem Commands
```

This is actually a very good way of entering comments into your batch files that has three advantages over standard "REM" lines:

- It is faster. DOS reads and processes REM lines, and that takes time. DOS just skips over lines that begin with a colon, unless it is looking for a line to GOTO. The increase in speed is more noticeable on slower machines.
- It handles pipes better. Because DOS reads and processes REM lines, it processes any pipes it finds. Create a remark line like "REM DIR>TEST" and you will find a 0-length file called TEST on your hard disk. (TEST is a 0-length file because the piping symbol told DOS to open a file for piping but the REM command did nothing to pipe to that file.) Because DOS skips lines beginning with a colon, pipes in those lines do not cause any problems.
- It handles command echoing better. REM lines are echoed to the screen when command echoing is on, while lines beginning with a colon are never echoed to the screen, so using a colon to start remark lines means they are never displayed when the batch file is running.

Bypassing the comments

The GOTO command and labels are detailed in Chapter 12. The GOTO command causes the batch file to jump over lines to a line beginning with a colon and the label specified after the GOTO command. For example, `GOTO START` would cause a batch file to jump to the line containing `:START`. Because jumping over lines is quicker than processing those lines, a batch file segment like this

```
GOTO START
REM Name:     BATCH.BAT
REM PURPOSE: Show Using Colon To Mark Documentation
REM VERSION: 1.00
REM DATE:     May 28, 1992
REM None Of These Lines Will Execute Or Show On The
REM Screen So They Function Similarly To Rem Commands
:START
```

would run quicker than the same batch file segment without the GOTO command. In fact, because the remark lines are jumped over and never executed, the REM command isn't needed, so this batch file could be written as

```
GOTO START
Name:     BATCH.BAT
PURPOSE: Show Using Colon To Mark Documentation
VERSION: 1.00
DATE:     May 28, 1992
None Of These Lines Will Execute Or Show On The
Screen So They Function Similarly To Rem Commands
:START
```

A warning about these tricks

Either of these tricks will give your batch file a slight speed boost; the more remarks you have grouped together and the slower your computer, the greater the speed boost. However, they are both nonstandard and make your batch files harder to understand. In addition, if you should later decide to compile your batch files with a compiler like Builder or Builder Lite (from TAB Books) then these tricks will have to be removed before the compiler will be able to compile the batch files. For these reasons, I don't recommend either approach.

Summary

Remark lines starting with the REM command make your batch files much easier to understand and follow.

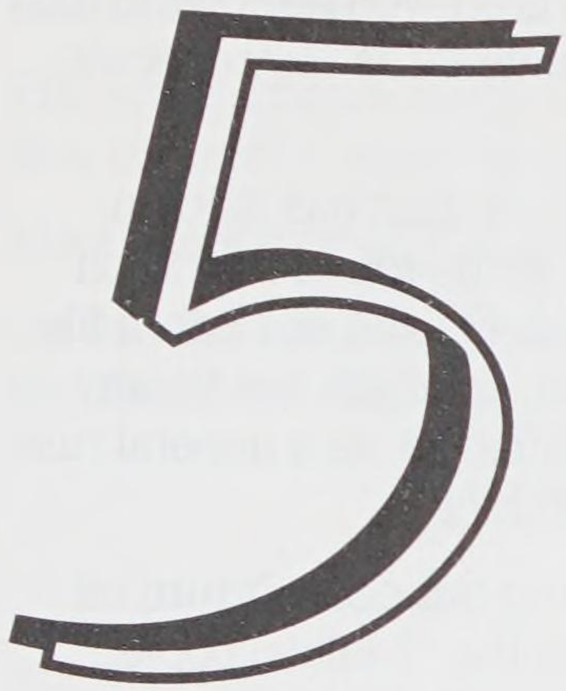
Written documentation is important to both the users of your batch files and anyone who later wants to modify the batch file.

Batch files should be able to explain their purpose to a user who starts them with a /? switch on the command line. How to do this is covered later in the book.

Adding some standard information to the top of each batch file is a good way to begin documenting your batch files and will allow batch files introduced later in this book to access that information in a useful fashion.

You can trick the operating system into skipping over documentation remarks quicker than with the REM command by preceding the remarks with a :* rather than the REM command. However, this is not recommended.

You can trick the operating system into skipping over documentation remarks quicker than with the REM command using a GOTO command to jump over the remarks. However, this is not recommended.



Let's keep it quiet

Let's take a moment to recreate the batch file called A.BAT we wrote earlier that performed a directory of the A drive. This time, let's use what we learned in the last chapter to include some documentation at the top of that batch file. Your resulting batch file might look like this:

```
REM NAME:    A.BAT
REM PURPOSE: Directory Of A Drive
REM VERSION: 1.00
REM DATE:    May 8, 1992
DIR A:
```

Now, run that batch file. Go ahead and do this now because the rest of the chapter will make more sense once you see this batch file run. We'll wait on you.

While the batch file runs properly and you get the information you want, the screen is very unattractive. As the batch file runs, every command in the batch file shows up on the screen—even when the user doesn't need or want this information.

Back in Chapter 1, I introduced the concept of a batch file being a script followed by the computer. Well, the computer has a bad habit of reading everything on the script. Take the following text, for example:

Billy Sally, you can't leave me. I need you.
Sally I'm leaving you. *[Opens door and walks out.]*

While the author expects Sally to open the door and walk out, he expects her to do it quietly. You also expect the computer to do what you tell it. But while running a tax program, you don't expect to see

```
Line Input A$
SalesTax = Income * .04 - (State * 2.3)
```

When you use a word processor or spreadsheet, you never see the code running. Why should a batch file be any different?

Command echoing

The process of showing commands on the screen while they're being executed is called *echoing* or *command echoing*. When this echoing is on, you see commands as they execute. When echoing is off, you don't see commands as they execute. Every batch file starts with echoing turned on.

You can turn off echoing in two different ways, and each method has its own uses. Normally, you can add an "ECHO OFF" command at the top of the batch file, which turns command echoing off for the entire batch file and any batch file this batch file runs (a process examined later in the book), although not for any other batch files that execute after this batch file terminates. So, as a general rule, you need an ECHO OFF command at the top of each batch file.

The ECHO OFF command turns off just command echoing but doesn't turn off DOS messages. If you copy a file in a batch file, you'll see the "1 File(s) copied" DOS message. If you use a batch file to start a program, you'll still see that program on the screen. All ECHO OFF affects is the display of batch file commands while that specific batch file is running.

☑ Remember: The ECHO OFF command keeps the screens produced by your batch files cleaner.

The second way to turn off command echoing only affects a single line. When a line in a batch file is preceded by an ampersand (@), that single line executes without being echoed to the screen, even if command echoing is on. Most users combine these two methods and add the command "@ECHO OFF" as the first line of their batch files.

We can now modify A.BAT to present a much clearer screen:

```
@ECHO OFF
REM NAME:    A.BAT
REM PURPOSE: Directory Of A Drive
REM VERSION: 1.00
REM DATE:    May 8, 1992
DIR A:
```

Only the first line has changed. Go ahead and modify your batch file and then run it. As you can see, with command echoing off, the screen is much clearer.

☑ Remember: Adding an ampersand in front of a command turns off command echoing for that command, even if command echoing is on for the entire batch file.

For almost all batch files, you can safely add an `@ECHO OFF` command to the top and forget about it. As we'll see in the next chapter, we can communicate with the user in spite of command echoing being off. Additionally, important DOS messages find their way to the screen in spite of command echoing being off. However, when we start working with ANSI in Chapter 21, some of the batch files we work with will need to have command echoing on in particular portions

of the batch file. Once command echoing is turned off, it can be turned back on with the `ECHO ON` command.

The normal state of batch files is command echoing on, where commands are displayed on the screen before being executed.

ECHO OFF turns off command echoing for an entire batch file and any other batch files that it runs.

Preceding a batch file command with an ampersand prevents command echoing for that command, regardless of the status of command echoing.

Most users start each batch file with an @ECHO OFF command.

On the rare occasion when you need command echoing on in a portion of a batch file, command echoing can be turned back on with ECHO ON.

Summary

Talking to the user

Some DOS commands are multi-modal, meaning that they perform different functions depending on how they are used. For example, the command SET by itself displays the contents of the environment, while the command

```
SET VARIABLE=VALUE
```

stores a value to an environmental variable. (If the value is missing, it deletes the environment variable—meaning that it stores a nul value in it.)

While SET is bi-modal, the PATH command is tri-modal because it has three different functions. The PATH command by itself displays the contents of the path, a command like

```
PATH=C:\;C:\WORD;C:\123
```

stores a value to the path and the command PATH; resets the path to nul.

☑ Remember: Some DOS commands perform several different functions depending on how they are used.

Like the PATH command, the ECHO command is tri-modal. In the last chapter, we saw that when it is combined with an ON or OFF, the ECHO command can be used to change the status of command echoing. This is only one use for the ECHO command. When used by itself, the ECHO command displays the status of command echoing.

Displaying messages

While the ECHO command can turn command echoing on or off and can display the status of command echoing, the ECHO command has another even more

important function: You can use it to communicate with the user. When ECHO is followed by a space and then a message, that message is displayed on the screen.

When command echoing is on, the message is actually displayed twice. It gets displayed once when the command is issued by the batch file (due to command echoing) and a second time when the command is carried out. When command echoing is off, the message is displayed only once.

Let's stop for a moment and look at a simple example. Key in this brief batch file segment and then run it:

```
@ECHO OFF
CLS
ECHO Hello World
ECHO I Am A Brand New Batch File!
```

When you run the file, the screen clears and the two messages are displayed, one per line.

You have several good reasons to echo messages in a batch file, many of which I'll mention here.

Telling the user what's going on

Any time a batch file is doing something for the user, you should let the user know what is going on—especially if the process takes a long time. Messages like "Defragmenting The Hard Disk: This Takes Two Hours" or "Backing Up To Tape Drive: This Takes Twenty Minutes" or even "Attempting To Log Onto Network: Please Wait" can go a long way to comfort the user when the computer doesn't immediately respond. After all, you don't want the user rebooting because she thinks that the computer has locked up.

Keeping the user informed

Not all commands take a very long time to run, but it's still nice to keep the user informed. For example, my AUTOEXEC.BAT file takes about a minute to run, but I've still got several messages in it to remind me what is going on. As an example, here is a shortened version of the file:

```
@ECHO OFF
REM NAME:    AUTOEXEC.BAT
REM VERSION: 4.00
REM DATE:    November 15, 1991
ECHO Setting Prompt and Environment
CALL C:\BAT\NICEPROM.BAT
CALL C:\BAT\SETS
CALL C:\BAT\SETPATH
ECHO Delete the .TMP Files In My Word Subdirectory
IF EXIST C:\WORD\*.TMP DEL C:\WORD\*.TMP
ECHO Loading CD ROM Driver
C:\CDROM\MSCDEX /D:MSCD210 /M:4 /L:F /V
C:\CDROM\MSCDINFO
ECHO Saving Hard Disk Information
IMAGE C: /SAVE
ECHO Turning NumLock Off
LOCKEYS N:OFF
ECHO Loading TSR Software
```

```
LOADHIGH FASTCONS
LOADHIGH DOSKEY /BUFSIZE=2500
SCREEN1
```

Most of the commands run software utilities. The lines beginning with CALL run other batch files. However, none of that is really important; what is important is all the ECHO commands. Some of these commands execute very quickly, while others take ten to fifteen seconds. Regardless, you can always look at the screen and know what the computer is doing at that moment.

Giving the user instructions

Sometimes the computer needs the user to either physically do something or provide some information. Messages displayed from within the batch file are an excellent way to provide the user with additional information on what to do.

An example will help you understand. The following batch file is an improved version of I.BAT for performing incremental backups:

```
@ECHO OFF
REM NAME:    I.BAT
REM PURPOSE: Perform Incremental Backups
REM VERSION: 2.00
REM DATE:    May 9, 1992
ECHO This Batch File Performs An Incremental Backup
ECHO Using The XCOPY Command
ECHO Insert A Freshly Formatted Disk In The A Drive
ECHO And Then Press Any Key When Ready
ECHO Note: If The Disk Fills Up, The XCOPY Command Will
ECHO Abort Without Finishing
ECHO If That Happens, Run The Batch File Over With
ECHO Another Freshly Formatted Disk To Copy The Files
ECHO That Were Missed The First Time
PAUSE
XCOPY C:\*.* /S /M A:
```

With the extra information provided by the ECHO commands, even someone who has never performed an incremental backup would know what to do.

☑ Remember: The ECHO command lets you display text on the screen to give instructions and keep the user informed.

Making attractive messages

Probably the single most important aspect of writing attractive messages is to make sure the message you echo to the screen is no wider than the screen. When a message is too long for the screen, it goes to 80 columns and then wraps to the next line without regard for words. If the break occurs in the middle of a word, then the line is broken in the middle of a word.

In addition to avoiding messages longer than 80 characters, I also tend to avoid long messages in general. It has been my experience that messages of 40 or fewer characters across are much easier for the user to read than messages that go all the way across the screen.

You can center titles by putting additional spaces between ECHO and the message. When I have several screens of information, I tend to display a title first on each screen and center that title with a blank line under it.

Where appropriate, I also use subtitles. For these, I display a blank line above and below the subtitle, and I underline the subtitle by displaying a series of dashes equal in length to the subtitle. Both items help to draw attention to the subtitle, as well as breaking up the 24 lines of text you can display on the screen.

Blank lines

As I just mentioned, I oftentimes use blank lines in my screen messages to call attention to certain portions of the message or just to break up long messages. However, displaying a blank line is not as simple as it sounds. You might expect that putting an ECHO on a line by itself would skip a line, but as I said earlier, this displays the status of command echoing.

You can choose from four different ways to display a blank line, with the one you select depending on several different factors.

Use an almost blank line The easiest way to display a blank line is to use commands like these:

```
ECHO !
ECHO .
```

Both of these commands display a printable character that is fairly inconspicuous. Most users will either not notice or else ignore the single inconspicuous character. This approach is the easiest but is also the least attractive.

Use an undocumented trick You can easily display a blank line in DOS using the command

```
ECHO.
```

where there is no space between the ECHO and the period. This is quick and easy, but it relies on an undocumented quirk in ECHO that might not be supported in future versions. This is best reserved for quick-and-easy batch files that are not likely to be used for too long.

Use a high-ordered ASCII character High-ordered ASCII characters are special characters supported by the computer but not shown on the keyboard. I'll explain more about them in Chapter 7. However, one of them is perfect for displaying blank lines. The high-ordered ASCII character with a value of 255 looks like a space on the screen, even though it isn't, so it's treated like any other character. Thus, if I represent this character with a <HO255>, the command

```
ECHO <HO255>
```

will display a blank line on the screen.

To create this high-ordered ASCII character, hold down the Alt key and enter 255 from the keypad with NumLock on. In Edit, press ^P first to tell it you will be entering a high-ordered ASCII character. Note that using the numbers at the top of the keyboard won't work. Because of the way you enter these characters, they are usually represented as Alt-#, or Alt-255 for this character. This is the notation I will use for the remainder of this book.

Some editors and word processors either do not accept high-ordered ASCII characters at all or make the process of entering them difficult. There's an easy solution to this problem. From the command line, enter the command

```
COPY CON 255 Alt-255^Z
```

to include an Alt-255 and nothing else in a file called 255. Now, import that into your word processor and paste it wherever you need it in the file.

Use a utility One of the files on the disk that comes with this book is a batch file utility called Batch Commander, written by Doug Amaral of hyperkinetix. One function of this program is to easily display a blank line on the screen, by use of the command

```
BATCMD SL
```

Note that you must include BATCMD.COM in your path somewhere. Full documentation for Batch Commander is provided in Chapter 25.

What you can & can't echo

While the ECHO command is very flexible, there are a few things that you either cannot display using the ECHO command or that displaying is more difficult than displaying most characters.

Pipes You can include the pipe symbols <, >>, >, or ¦ in an ECHO statement. However, if you do, the text to be displayed by the ECHO command is piped into a file or alternative DOS device rather than being displayed on the screen. Piping is examined in more detail in Chapter 15.

☠ Warning: Extreme care should be used when including piping symbols in an ECHO command. The text is piped somewhere rather than displayed on the screen.

Percent sign You cannot directly include a percent sign in an ECHO statement because DOS tries to interpret a single percent sign as a variable. If you need a percent sign in your ECHO statement, include a double percent sign (%%). DOS will display this as a single percent sign.

☑ Remember: To display a single percent sign in an ECHO command, the batch file must have two percent signs.

On/Off You can't echo any message that begins with an "on" or "off" because DOS interprets this as an ECHO ON or ECHO OFF command. You can solve this in two ways. The easiest way is to rewrite the message so that the words "ON" or "OFF" are not the first words in the message. If rewriting the message isn't possible, then begin the message with the Alt-255 "space" described. Because the high-ordered Alt-255 is treated as a character, the "ON" or "OFF" are no longer the first words in the message.

Longer fancier messages

When you must display a lot of text or when you want fancier formatting, you have three possibilities:

- ❑ Place the text in a separate file and use the TYPE command to send it to the screen. See Chapter 20 for more detail.
- ❑ Use ANSI commands to improve the appearance of the text (examined in Chapter 21).
- ❑ Use a screen compiler to turn each screen full of text into a stand-alone program. (See Chapter 20.)

Each of these approaches is detailed later on in the book, once you have more experience with the simpler batch file commands.

Summary

Some DOS commands have two or more functions, depending on how they are used.

The ECHO command can turn on/off command echoing, display the status of command echoing, and display a message on the screen—all depending on how it is used.

Displaying messages to the user via the ECHO command is useful to tell the user that the batch file is running a process that takes a very long time.

Displaying messages to the user via the ECHO command is useful to tell the user what the batch file is doing.

Displaying messages to the user via the ECHO command is useful to give the user additional instructions.

Echo messages should be 80 characters or less to prevent unexpected wrapping on the screen.

Shorter echo messages are easier to read than longer messages.

Echo messages can be centered by adding additional spaces between ECHO and the message.

Nearly blank lines can be displayed by printing a single inconspicuous character on a line.

A blank line can be displayed with the undocumented command of placing a period directly after ECHO without a space in between.

A blank-looking line can be displayed by echoing the high-ordered ASCII character Alt-255.

The Batch Commander program included with this book includes the SL (skip line) option to display a blank line.

Piping symbols cannot be displayed in an ECHO command.

In order to display a single percent sign, the ECHO command must have two percent signs side by side.

The words "ON" and "OFF" cannot be used as the first word of an ECHO message.

7 Using high-ordered ASCII characters

DOS can use 256 different characters, each one having an associated ASCII value that runs from 0 to 255. Figure 7-1 shows an ASCII chart listing each value. Most users only concern themselves with the values below 128 because these are the values that can be entered by pressing a key on the keyboard. However, the values of 128 and higher offer some nice formatting tools for batch files.

Standard ASCII chart (character codes 0–127)

Code	Function	Character	Code	Function	Character
000	NUL	(none)	016	DLE	►
001	SOH	☺	017	DC1	◄
002	STX	☻	018	DC2	↕
003	ETX	♥	019	DC3	!!
004	EOT	♦	020	DC4	¶
005	ENQ	♣	021	NAK	§
006	ACK	♠	022	SYN	-
007	BEL	•	023	ETB	↨
008	BS	◘	024	CAN	↑
009	TAB	○	025	EM	↓
010	LF	◙	026	EOF	→
011	VT	♂	027	ESC	
012	NP	♀	028	FS	└
013	CR	♪	029	GS	↔
014	SO	♫	030	RS	▲
015	SI	☼	031	US	▼

7-1
An ASCII chart.

Standard ASCII characters

Code	Character	Code	Character
032	sp	080	P
033	!	081	Q
034	“	082	R
035	#	083	S
036	$	084	T
037	%	085	U
038	&	086	V
039	’	087	W
040	(	088	X
041	)	089	Y
042	*	090	Z
043	+	091	[
044	’	092	\
045	–	093	]
046	.	094	^
047	/	095	_
048	0	096	`
049	1	097	a
050	2	098	b
051	3	099	c
052	4	100	d
053	5	101	e
054	6	102	f
055	7	103	g
056	8	104	h
057	9	105	i
058	:	106	j
059	;	107	k
060	<	108	l
061	=	109	m
062	>	110	n
063	?	111	o
064	@	112	p
065	A	113	q
066	B	114	r
067	C	115	s
068	D	116	t
069	E	117	u
070	F	118	v
071	G	119	w
072	H	120	x
073	I	121	y
074	J	122	z
075	K	123	{
076	L	124	|
077	M	125	}
078	N	126	
079	O	127	(bs)

Extended ASCII chart (character codes 128–255)

Code	Character	Code	Character
128	Ç	174	«
129	ü	175	»
130	é	176	▒
131	â	177	▒
132	ä	178	▓
133	à	179	│
134	å	180	┤
135	ç	181	╡
136	ê	182	╢
137	ë	183	╖
138	è	184	╕
139	ï	185	╣
140	î	186	║
141	ì	187	╗
142	Ä	188	╝
143	Å	189	╜
144	É	190	╛
145	æ	191	┐
146	Æ	192	└
147	ô	193	┴
148	ö	194	┬
149	ò	195	├
150	û	196	─
151	ù	197	┼
152	ÿ	198	╞
153	Ö	199	╟
154	Ü	200	╚
155	¢	201	╔
156	£	202	╩
157	¥	203	╦
158	₧	204	╠
159	*f*	205	═
160	á	206	╬
161	í	207	╧
162	ó	208	╨
163	ú	209	╤
164	ñ	210	╥
165	Ñ	211	╙
166	ª	212	╘
167	º	213	╒
168	¿	214	╓
169	⌐	215	╫
170	¬	216	╪
171	½	217	┘
172	¼	218	┌
173	¡	219	▮

7-1

Continued.

220	▄	238	∈
221	▌	239	
222	▐	240	
223	▀	241	±
224	α	242	≥
225	β	243	≤
226	Γ	244	⌠
227	π	245	⌡
228	Σ	246	÷
229	σ	247	≈
230	µ	248	°
231	τ	249	∙
232	Φ	250	·
233	Θ	251	√
234	Ω	252	^{n}
235	δ	253	^{2}
236	∞	254	■
237	φ	255	(null)

If you wanted to draw a box without the high-ordered ASCII characters, you would have to use colons for the pipe symbol (remember, the ¦ symbol is a pipe and cannot be used in ECHO messages), the plus sign for the corners, and the dash for the horizontal bars. However, with the high-ordered ASCII characters, nice boxes are easy.

HI-ASCII.BAT in FIG. 7-2 illustrates this. The lines that appear to be an ECHO by itself also have an Alt-255 character to produce a blank line. A copy of HI-ASCII.BAT is on the disk, and you can run it to see how the boxes look on your screen.

Characters with an ASCII value of 128 or higher are called high-ordered ASCII characters. Figure 7-3 lists those high-ordered ASCII characters that are most useful in batch files.

If you ever need to quickly look up an ASCII value while running DOS, you don't have to look for your copy of this book. Just copy ASCII.EXE from the enclosed disk to a subdirectory in your path, and then enter `ASCII` at the command line prompt to get a display of all the ASCII characters with values 30–255. (Values below 30 are mostly used for control functions, so most of them won't display on the screen anyway.) ASCII.EXE requires very little memory so chances are good you can run it while shelled out from your favorite editor or word processor.

If you are using the COPY command to create your batch files, then entering high-ordered ASCII characters is easy. Take these steps:

Entering high-ordered ASCII characters

1. Make sure the NumLock is on.
2. Hold down the Alt key.
3. Type the three-digit code for the character using the number pad. (The numbers at the top of the keyboard will not work.)
4. Release the Alt key.
5. Repeats Steps 2–4 for any additional characters you want to enter.

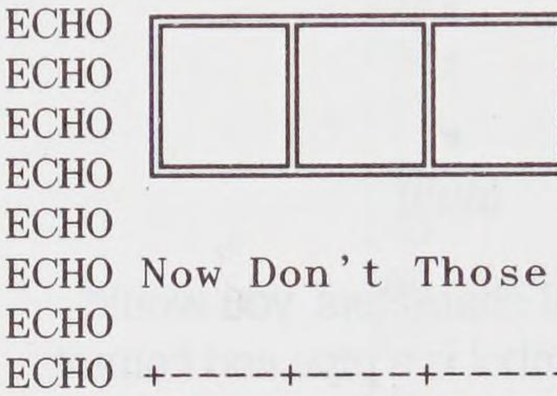
This method is not limited to just high-ordered ASCII characters; you could use Alt-68 to enter a capital D, for example. However, using the keyboard for the low-

```
@ECHO OFF
REM NAME:     HI-ASCII.BAT
REM PURPOSE: High-Ordered ASCII Demonstration
REM VERSION: 1.00
REM DATE:     May 10, 1992
CLS
ECHO
ECHO
ECHO   High Order ASCII Character Demonstration
ECHO
ECHO
ECHO
ECHO
ECHO
ECHO
ECHO
ECHO
ECHO
ECHO
ECHO
ECHO
ECHO Now Don't Those Look Better Than This
ECHO
ECHO +----+----+----+----+----+----+----+----+----+----+
ECHO :    :    :    :    :    :    :    :    :    :    :
ECHO :    :    :    :    :    :    :    :    :    :    :
ECHO +----+----+----+----+----+----+----+----+----+----+
PAUSE
```

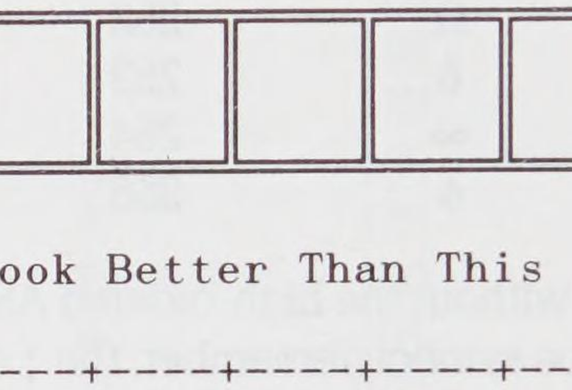
7-2

HI-ASCII.BAT shows how high-ordered ASCII characters can produce attractive boxes in a batch file.

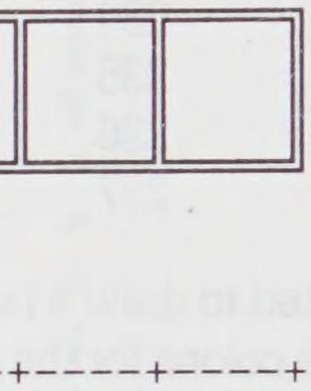
In Context



Double Lines



Single Lines

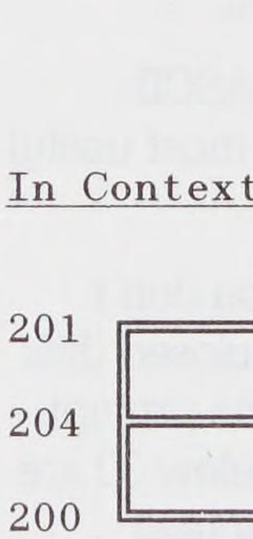


Mixed Lines

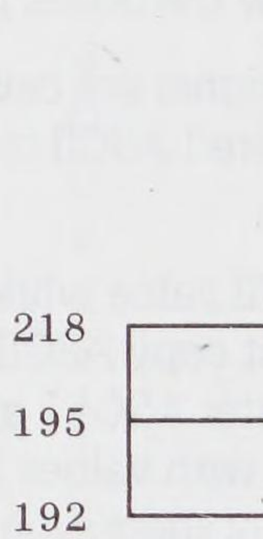
7-3

These high-ordered ASCII characters are particularly useful in formatting batch files.

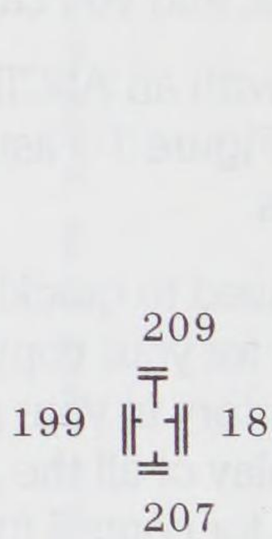
In ASCII Order



Double Lines



Single Lines



Mixed Lines

ordered ASCII characters is much easier. The process is almost identical with the Edit editor, you just press ^P before entering the Alt-sequence.

Some editors make it easy to enter the box-drawing characters. Microsoft Word, for example, has a line-drawing utility that draws using single- or double-lines as you move around the screen and automatically inserts the proper joint when the lines cross.

Many word processors make it difficult to enter high-ordered ASCII characters. Microsoft Word, for example, uses the Alt-key to pull down its menus so you sometimes have to enter the high-ordered ASCII character several times before Word figures out you are entering a character and not requesting a menu. Some versions of WordStar require you to enter a ^P and then the three-digit code rather than using the Alt-key combination. If the Alt-key combination doesn't work in your editor or word processor, check your manual to see what your program expects.

What if your editor won't let you?

A few editors and word processors make it difficult or impossible to enter high-ordered ASCII characters into your batch files. There is an easy solution that works with almost every editor and word processor. Take these steps:

1. From the command line prompt, enter the command `COPY CON 255`, only replace the "255" with the name of the file to use. I found it handy to create one file per character for each character I use a lot and to use the ASCII number as the filename.
2. Hold down the Alt key and type in the ASCII code for the high-ordered ASCII character to include in the file.
3. Press F6.
4. Press Enter.

At this point, the file exists and contains the character of interest. Now, to get this character into your file, just import it into the file using the editor's/word processor's import command. If it does not have one, open this file first and copy the character to the "paste buffer" (the area where you store characters to paste into other files). If your editor or word processor cannot import a file and doesn't have a paste buffer, you might want to consider finding another editor.

Summary

High-ordered ASCII characters make it easy to draw attractive boxes inside a batch file.

ASCII.EXE on the enclosed disk displays an ASCII chart of most characters on the screen.

Many editors let you enter high-ordered ASCII characters by holding down the Alt key and entering the associated number from the number pad.

If your editor or word processor will not let you enter high-ordered ASCII characters, you can pipe them to a file using DOS and then read that file into your editor or word processor.

Hold that batch file!

Imagine you are running the following batch file segment:

```
@ECHO OFF
CLS
ECHO Warning: The Following Program Will Do Damage
ECHO.
ECHO          It Erases All Documents
ECHO          It Damages .EXE Programs In This
ECHO          Subdirectory
ECHO          And It Removes The \BAT Subdirectory
ECHO.
RUNIT
```

This batch file presents a fairly stern warning to the user but doesn't give the user the ability to do anything about it: by the time the user has read the warning, the program has already started running!

It is unlikely you would ever run such a damaging program—at least intentionally. However, the purpose of a warning is to give the user the time to read it and react to it.

Additionally, what if you had a ton of information to give to the user and that information was going to take sixty ECHO commands? Using the commands we have learned so far, all you could do is put one after another in a batch file. However, unless the computer was very slow, the messages would scroll by too fast for the user to read.

Pausing the batch file

The PAUSE command is the solution to this dilemma. When a batch file comes to a PAUSE, it stops and displays a "Press any key to continue..." message. The batch file then waits until the user presses any printable key. Keystrokes like Alt, Shift, Ctrl, NumLock, or others that don't produce a character on the screen won't cause a batch file to resume while waiting on a PAUSE command.

If the user presses Ctrl-Break, the batch file will respond with a "Terminate batch job (Y/N)?" message. Using this, you can build an "escape clause" into your batch files. Let's reconsider the first batch file with a PAUSE command used as an escape clause:

```
@ECHO OFF
CLS
ECHO Warning: The Following Program Will Do Damage
ECHO.
ECHO          It Erases All Documents
ECHO          It Damages .EXE Programs In This
ECHO          Subdirectory
ECHO          And It Removes The \BAT Subdirectory
ECHO.
ECHO To Skip Running The Program, Hold Down The Control
ECHO Key And Press The Break Key
ECHO When The Computer Asks "Terminate batch job
ECHO (Y/N)?" Respond With A Y
PAUSE
RUNIT
```

The batch file now tells the user how to abort processing and has a PAUSE and that gives the user a chance to think about the messages on the screen and decide how to respond.

Remember: The PAUSE command stops the computer while the user reads the screen or thinks about a response. It even gives the user a chance to stop the batch file by pressing Ctrl-Break.

PAUSE is useful even if the user won't need to press Ctrl-Break; it simply gives the user time to read the messages and think about them before continuing. If you are going to display several different screens of information, then PAUSE followed on the next line by CLS is critical to ensure that the user is able to read all the information.

Aborting the batch file

As mentioned previously, the Ctrl-Break keystroke can be used to stop a batch file immediately and display a "Terminate batch job (Y/N)?" message. If the user responds with a y, control is immediately returned to DOS. If the user responds with an n, control returns to the batch file.

This makes it easy for the user to abort a batch file prior to running a dangerous command. Later on, in Chapter 19, we will see a better approach to giving the user an escape clause.

Summary

PAUSE stops a batch file until the user presses any printable keystroke.

If the user presses Ctrl-Break at a PAUSE command—or any other point in the batch file—DOS gives the user the option to stop processing of the batch file.

9 Applications and problems

It's time to stop and catch our breath. While it might not seem so at first, we have covered a tremendous amount of material. You now know how to use a brand new editor, and you've learned how to write simple batch files that contain a mixture of DOS commands and a few batch subcommands.

That's why we should take a few moments and make sure that you're up to speed before we continue. What follows are three quick problems. Take a few moments and try to work them. If you get stuck, glance back through the first few chapters. It's important that you have the skills necessary to solve these problems before you go on; otherwise, you'll find continuing difficult.

Problem 1

Write a batch file to format a floppy disk that is in the A drive. Assume that FORMAT.COM has been renamed to XYZ.COM. Use plenty of ECHO statements to tell the user what to do, and be sure to turn command echoing off so that the user won't see the XYZ command and be tempted to run the program directly. Once the disk is formatted, restart the menu program with the command `MENU`. You can assume that XYZ.COM is in the path. Also, you should make sure to be in the root directory of the C drive sometime prior to restarting the menu.

Problem 2

Write an AUTOEXEC.BAT file that performs the following tasks:

1. Sets the path to include the following subdirectories:

```
C:\
C:\MSDOS
C:\NORTON
C:\123
C:\WP
```

```
C:\UTILITY
C:\BAT
C:\MENU
```

2. Creates the following environmental variables:

```
TEMP=C:\TEMP
DOSONLY=YES
LIB=D:\BLDLITE
OBJ=D:\BLDLITE
```

3. Runs a program called Noprtscr in the C:\UTILITY subdirectory to turn off screen printing.
4. Starts a menu program in the C:\MENU subdirectory using the command MENU.

While order generally doesn't matter, remember that if the path doesn't yet point to a subdirectory, then you must change to that subdirectory before running a program; also, loading the menu program must be the last command in the batch file.

Problem 3

Your system has three accounting programs—DAILY.EXE, WEEKLY.COM and ANNUAL.EXE—that can only be run on your computer by authorized people. Write an informational batch file called ACCTHELP.BAT to give the users information about these programs. That information should include the following things:

- The names of the programs.
- The people authorized to run the programs.
- The fact that passwords are required.
- The name of the person to contact for help with these programs.

Finally, note how important it is to perform a backup after running one of these programs and how a copy of the backup must be sent to the appropriate department at the corporate headquarters. For the names and numbers you need for this problem, make up any information that you don't have.

Answers

You have a number of ways to tackle each of these problems. My solutions are shown in Appendix A in the back of the book. It isn't important that you match my answers; if you developed a working solution to the problem, your solution is just as good as mine and maybe even better.

Part three

Intermediate batch file topics

Now that you've seen the simple batch file commands, it's time to take a look at some slightly more complex commands. As it turns out, the chapters in this section are the "meat" of the book. As you write your own batch files, you'll find that the vast majority of the batch file commands you use are either the REM or ECHO command or the commands covered in these chapters. Just to be sure, there is a chapter at the end of this section where you can test your knowledge.

Replaceable parameters Unless you have a way to get information into a batch file, it is difficult to have the batch file perform differently each time it runs. Replaceable parameters are a method of adding additional information on the command line that the batch file then has access to. This information can be used as file names, extensions, subdirectories, commands, or just about anything you use in a batch file. — **Chapter 10**

Using more than ten replaceable parameters When it first starts, the batch file only has access to its name and the first nine pieces of information you entered on the command line. This chapter shows you how to gain access to additional information. — **Chapter 11**

Looping and program flow This chapter shows you how to define different execution paths through the batch file and how to create loops within a batch file. — **Chapter 12**

The environment The environment is an important scratch pad for the operating system. It uses the environment to store information that all the programs need access to. — **Chapter 13**

Errorlevel The errorlevel is a one-byte portion of memory that programs can use to communicate to each other or with a batch file. While only a few DOS programs use the errorlevel, most batch file utilities use it to pass results back to the batch file that ran them. This chapter provides an overview of the Errorlevel. — **Chapter 14**

DOS punctuation Many of the batch files you have seen so far have used varying types of DOS punctuation to help them perform their work. This chapter documents that punctuation as a resource to help you understand these batch files as well as write more powerful batch files of your own. — **Chapter 15**

Logic testing The different paths you learned to create in Chapter 12 are useless unless you have a method of selecting different paths at different times. — **Chapter 16**

Additionally, the loops you learned to create in Chapter 12 are useless unless you have a way to get out of them. In both cases, logic testing is the solution. This chapter shows you how to build logic testing into your batch files. Logic testing is what turns a batch file away from being just a dumb script the computer plays back into an intelligent program.

Chapter 17 **Applications and problems** Before moving on to the advanced batch subcommands, you should understand the commands and concepts covered so far. This chapter is a quick test. Pass it, and you're ready to move on.

10 Replaceable parameters

Batch files would be useful if they did nothing more than what's been covered so far. However, one big limitation of the commands that we've looked at is that they're "written in stone." If you want to erase a file with a batch file that just uses these commands, you'll need to physically enter that filename in the batch file code. This is called hardwiring.

Sometimes you want to hardwire filenames. If you want to copy all your *.WK1 spreadsheet files to a floppy disk as backup, then hardwiring the `COPY *.WK1 A:` command is appropriate. Other times, however, you want the flexibility to have the batch file process different files. Replaceable parameters offer just such flexibility.

You need to know two things in order to use replaceable parameters: how to give the values for the replaceable parameters to the batch file, and how to code the replaceable parameters in the batch file.

Sending replaceable parameters to a batch file

Giving the values of the replaceable parameters to the batch file is easy. Replaceable parameters are entered after the batch file name and before pressing Enter. For example, the command

```
BATCH Parameter_1 Parameter_2 Parameter_3 Parameter_4
```

will run BATCH.BAT and pass it *Parameter_1* through *Parameter_4* as replaceable parameters.

Each parameter is separated by either a space, comma, or semicolon. How many replaceable parameters does the previous line have? The obvious (and wrong) answer is four. DOS counts the name of the batch file as a parameter, so there are five replaceable parameters on this line. Just to make things confusing, DOS names

them %0, %1, %2, %3, and %4 and not the %1 through %5 you'd expect. (See FIG. 10-1.)

%0 = BATCH
%1 = *.BAK
%2 = *.DOC
%3 = *.TMP
%4 = *.WKS
%5 = *.WK1
%6 = *.TXT
%7 = *.DBF
%8 = *.PRG
%9 = *.IN

10-1
*When the command line is BATCH *.BAK *.DOC *.TMP *.WKS *.WK1 *.TXT *.DBF *.PRG *.IN, these are the replaceable parameters as seen by BATCH.BLD.*

The notation DOS uses leaves me with a dilemma in writing this book. Do I refer to %1 as the first replaceable parameter or the second? It truly is the second; however, that causes confusion for two reasons. First, most readers associate second with 2 and not 1, so they expect the second replaceable parameter to be %2. Second, %1 is actually the first piece of information you enter for the batch file to use, so you really intended it as the first replaceable parameter; you just had to enter the name of the batch file first in order to run it. Thus, I've decided that, throughout this book, I will refer to %1 as the first replaceable parameter, %2 as the second, %3 as the third, and so on. When I need to talk about %0, I will refer to it by the name %0.

Note: When describing replaceable parameters, I will refer to %1 as the first replaceable parameter, %2 as the second, and so on. When I want to talk about %0, I will refer to it by the name %0.

The general rule is that everything you enter on the command line prior to the space becomes %0, everything between the first and second spaces becomes %1, and so on. This means that the name of the batch file becomes %0. This is stored exactly as you enter it on the command line, so if you run BATCH.BAT with the command `BATCH`, %0 contains "BATCH" with that capitalization and without the .BAT extension. If you run it with the command `BATCH.BAT`, then %0 contains "BATCH.BAT" with the extension—just as you entered it.

When you boot your computer, DOS automatically runs the AUTOEXEC.BAT file in the root directory without the name being entered on the command line. When AUTOEXEC.BAT is run in this fashion, the %0 variable is equal to nul. Note that some versions of MS- and DR DOS have problems when the AUTOEXEC.BAT file is

being run during a boot if you test on the %0 variable so use this fact with caution. However, when one batch file runs another batch file with the CALL command, the %0 variable for that second batch file is its name, even though its name is not entered on a command line.

However, there is an exception to this rule of %0 containing everything you entered before the first space exactly as you entered it. Many programs expect command line parameters to be entered with a slash, like the command:

```
DIR/W
```

DOS doesn't care if you place a space between the DIR and the /W: it knows that DIR is the command and the /W is a parameter. DR DOS handles batch files the same way. If you run BATCH.BAT with the command

```
BATCH/H
```

then BATCH becomes %0 and /H becomes %1. However, MS DOS works differently and treats everything until the first space as %0. This can make writing batch files to run under either operating system difficult. We will revisit this issue when we look at logic testing.

Warning: DR DOS and MS-DOS handle replaceable parameters differently when the name of a batch file is followed immediately by a slash with information after the slash. DR DOS makes the slash and everything that follows %1. MS-DOS treats everything before the first space as %0. This makes it very difficult to write batch files that run under both MS- and DR DOS, and that accept parameters with a slash.

This can be quite confusing so let's look at a couple of examples. For each of these examples, we will need a batch file. SHOWREPL.BAT is shown here and handles the job nicely:

```
@ECHO OFF
REM NAME:    SHOWREPL.BAT
REM PURPOSE: Show Replaceable Parameters
REM VERSION: 1.00
REM DATE:    May 12, 1992
ECHO %%0 = %0
ECHO %%1 = %1
ECHO %%2 = %2
ECHO %%3 = %3
ECHO %%4 = %4
ECHO %%5 = %5
ECHO %%6 = %6
ECHO %%7 = %7
ECHO %%8 = %8
ECHO %%9 = %9
```

Before running SHOWREPL.BAT, you should note that the ECHO commands with the %%0 are used to display a %0. In order to display a percent sign using an ECHO command, DOS requires the ECHO to have two percentage signs.

SHOWREPL.BAT is expanded later, and that expanded version is included on the disk. However, that expanded version works fine for these examples, so you don't need to key in the batch file. If you run SHOWREPL.BAT with the command

```
SHOWREPL/H/I/J One
```

then the replaceable parameters are

```
%0  SHOWREPL/H/I/J
%1  One
```

Note that the three switches /H/I/J are not separated from the SHOWREPL. DOS does this consistently. When SHOWREPL.BAT is started with the command

```
SHOWREPL /H /I /J One
```

then the replaceable parameters are

```
%0  SHOWREPL
%1  /H
%2  /I
%3  /J
%4  One
```

just as you would expect.

If you are still confused, you might want to run SHOWREPL.BAT several times with different combinations of replaceable parameters to make sure you understand how DOS is translating your command lines into replaceable parameters. A copy of SHOWREPL.BAT is included on the enclosed disk for that purpose.

Using replaceable parameters in your batch files

Replaceable parameters are global variables and are available to any command in the batch file. However, once the batch file terminates, the values of the replaceable parameters are terminated as well. When a batch file runs a second batch file, the replaceable parameters are not available to that second batch file unless they are specifically passed to that second batch file by the command that runs it. This is covered in the chapter on subroutines. However, once the second batch file terminates and control returns to the original batch file, its replaceable parameters are once again available to it.

DOS uses replaceable parameters as if they don't exist. That sounds like an extreme statement, but the supporting logic is solid. Everywhere *you* see a replaceable parameter, *DOS* sees the value of that parameter. That statement will make more sense if we look at an example.

First, we need a batch file. I've modified the SHOWREPL.BAT batch file we used earlier to include some REM commands at the bottom of the batch file along with a PAUSE command.

```
@ECHO OFF
REM NAME:    SHOWREPL.BAT
REM PURPOSE: Show Replaceable Parameters
REM VERSION: 1.00
REM DATE:    May 12, 1992
ECHO %%0 = %0
ECHO %%1 = %1
ECHO %%2 = %2
ECHO %%3 = %3
ECHO %%4 = %4
ECHO %%5 = %5
```

```
ECHO %%6 = %6
ECHO %%7 = %7
ECHO %%8 = %8
ECHO %%9 = %9
ECHO
ECHO Now, Watch These Remark Statements
ECHO After I Turn Command Echoing On
ECHO ON
REM This Batch File Shows The Values
REM For The First Ten Replaceable
REM Parameters (%0-%9) Entered On
REM The Command Line
@PAUSE
```

Before running SHOWREPL.BAT, you should note that the next to last REM line has two replaceable parameters in it. Because command echoing is on when the batch file reaches this point, we will want to notice how DOS treats this line. Go ahead and run the batch file. Be sure to enter some replaceable parameters when you run it.

Notice that, when a batch file executes with command echoing on—beginning in the middle of the batch file—each command in the batch file is preceded by the command line prompt. In other words, DOS treats the commands in the batch file exactly like it would if you entered them from the command line prompt. This is consistent with our idea of a batch file being a script the computer follows.

Now, notice the next to last REM line. In the batch file, it reads `REM Parameters (%0-%9) Entered On`. When I ran this batch file, I ran it with the command:

```
showrepl One 2 Three Four 5 6,7,8 Nine
```

so `showrepl` became %0 and `Nine` became %9. If you do the same, in the REM line on the screen, you can see that DOS has replaced the "%0–%9" in the batch file with "showrepl-Nine."

With the single exception of a %%0, everywhere there's a %0 in the batch file, DOS sees the value of the first replaceable parameter. Except for %%1, everywhere there's a %1 in the batch file, DOS sees the value of the second replaceable parameter, and so on. In fact, they are called replaceable parameters for exactly that reason. Because the values come from the command line, you can replace them with new values each time you run the batch file.

Limits on replaceable parameters

DOS does not directly place a limit on the number of replaceable parameters you can specify on the command line. However, it does place an indirect limit. A command line is limited to 127 characters, and the name of the batch file and the first replaceable parameter—as well as each set of replaceable parameters—must be separated by a space, comma or semi-colon. If you run a batch file with a single character name, like A.BAT, and you give it only single character replaceable parameters, like:

```
A 1 2 3 4 5 6 7 8 9 0 A B C ...
```

then you will have room for 63 replaceable parameters. However, only the replaceable parameters %0–%9 are valid in your batch file. We will see a way around this problem in Chapter 11.

Uses for replaceable parameters

Back in Chapter 1, we looked at the following batch file segment to start Microsoft Word:

```
C:
CD\WORD
WORD
CD\
MENU
```

At the time, we didn't give it a name, so let's call it WP.BAT.

Like many programs, Microsoft Word gives you the ability to immediately begin editing a file by specifying the name of that file on the command line. If you start Word with the command

```
WORD CHAP-10
```

you not only start Word, you cause it to load CHAP-10.DOC so that you can begin to edit it. (If CHAP-10.DOC doesn't yet exist, Word will warn you and then create the document for you.) Using replaceable parameters and the commands we have learned since Chapter 1, we can make this a much more useful batch file:

```
@ECHO OFF
REM NAME:    WP.BAT
REM PURPOSE: Run Microsoft Word
REM VERSION: 2.00
REM DATE:    May 12, 1992
C:
CD\WORD
WORD %1
CD\
MENU
```

Now, when the user starts this batch file with the command:

```
WP CHAP-10
```

the batch file both starts Word and tells Word to load the CHAP-10.DOC document. (Note that Word doesn't require you to enter the extension; it automatically uses .DOC.)

You might wonder what happens when you start WP.BAT without a replaceable parameter. In that case, %1 has no value. DOS performs an exact replacement of the value of %1, so the batch file line `WORD %1` is treated by DOS as "WORD."

In other words, no replaceable parameter was entered so DOS treats that line as though the %1 does not exist. That gives you the flexibility of using the same batch file both when you want to specify a file name and when you don't.

Replaceable parameters as commands

DOS is extremely literal in replacing the replaceable parameters with their values and treating those values as though they were hardwired in the batch file. So literal, in fact, that replaceable parameters can even be used as commands. For example, this batch file

```
@ECHO OFF
REM NAME:    COMMANDS.BAT
REM PURPOSE: Show Using Replaceable Parameters
```

```
REM              As Commands
REM VERSION:     1.00
REM DATE:        May 12, 1992
%1
%2
%3
```

would treat the first three replaceable parameters as commands. If you ran the batch file with the command:

```
COMMANDS DIR/W CHKDSK/F MENU
```

it would first perform a DIR/W, then run CHKDSK/F, and finally run the menu program. Unlike the FOR command, the slash is not used as a divider for replaceable parameters, so these commands work as expected.

Of course, just because this works doesn't mean that it is a good idea. When you allow the user to enter the command, you lose the control the batch file was designed to provide. So, as a general rule, using replaceable parameters as commands is not a good idea.

A couple of examples

Let's create a couple of example batch files using replaceable parameters. As I introduce additional commands for working with replaceable parameters as well as general batch file commands, we will revisit these batch files to make improvements to them. They are on the disk that comes with this book—but only in their final form. So, if you want to experiment with the intermediate versions, you will need to enter them yourself.

This first batch file—A.BAT—performs a directory of the A drive. It uses a single replaceable parameter. When you run it without a replaceable parameter, it gives you all the files on the A drive. You can enter a replaceable parameter like "*.DOC" to narrow the search.

```
@ECHO OFF
REM NAME:        A.BAT
REM PURPOSE:     Directory Of A Drive
REM VERSION:     1.00
REM DATE:        May 12, 1992
DIR A:%1 /P
```

This second batch file—RONNYMD.BAT or Ronny's MD command—will create a subdirectory branching off the current subdirectory and then change to that subdirectory.

```
@ECHO OFF
REM NAME:        RONNYMD.BAT
REM PURPOSE:     Make And Change To A Subdirectory
REM VERSION:     1.00
REM DATE:        May 12, 1992
MD %1
CD %1
```

A.BAT runs fine if you don't enter a replaceable parameter—you just get a listing of all the files. RONNYMD.BAT has a small problem. The MD %1 becomes MD when no replaceable parameter is entered—resulting in a DOS "Required parameter

missing" error message. Without a replaceable parameter, the "CD" command just displays the current subdirectory. One of the enhancements we will be making to these batch files in the chapters that follow is error-checking to avoid problems.

While A.BAT runs fine without a replaceable parameter and RONNYMD.BAT has a problem with one line, this final batch file—TOA.BAT—has significant problems when used with the wrong number of replaceable parameters.

```
@ECHO OFF
REM NAME:     TOA.BAT
REM PURPOSE: Copy .CMP And .DOC Files To A Drive
REM          While Deleting .BAK Files
REM VERSION: 1.00
REM DATE:     May 12, 1992
DEL %1.BAK
COPY %1.DOC A:
COPY %1.CMP A:
REM %2 - %8 Are Handled In A Similar Fashion
REM I've Removed Their Code From The Figure
REM For The Sake Of Space
DEL %9.BAK
COPY %9.DOC A:
COPY %9.CMP A:
```

TOA.BAT is designed to be run followed by exactly nine replaceable parameters. Each replaceable parameter designates a document to be copied to the A drive and must be entered without the .DOC extension. The batch file performs three steps for each replaceable parameter:

1. Deletes the .BAK backup file.
2. Copies the .DOC document file to the A drive.
3. Copies the .CMP user dictionary for that file to the A drive.

As written, this batch file has several significant problems. In fact, I've included it because it has many of the problems associated with using replaceable parameters in batch files. The problems with this batch file are as follows:

- TOA.BAT doesn't check to make sure that nine replaceable parameters were entered. If only six were entered, then %7, %8, and %9 don't exist. For each of these, the `DEL .BAK`, `COPY .DOC A:`, and `COPY .CMP A:` commands generate a DOS "Invalid directory specified" error message. The filename is missing because the replaceable parameter is empty, and DOS treats the period before the extension as part of a path.
- TOA.BAT ignores replaceable parameters beyond %9.
- TOA.BAT doesn't check to see if the .BAK file exists before it tries to delete it.
- TOA.BAT doesn't check to see if the .DOC and .CMP files exist before it tries to copy them.

In the chapters to come, we will see how to avoid each of these problems.

Summary

A replaceable parameter is information you enter on the command line after the name of a batch file when you run that batch file.

Replaceable parameters must be separated by a space, comma, or semicolon on the command line.

Replaceable parameters are global variables that are available anywhere in the batch file.

In addition to being available anywhere in a batch file, replaceable parameters can be passed by one batch file to another batch file or by a batch file to a program.

Replaceable parameters are used in a batch file using the terms %0–%9. The %0 is the name of the batch file, %1 is the first replaceable parameter, %2 is the second, and so on.

When a batch file name on the command line is followed by a slash and switches, and there is no space between the name of the batch file and the slash, MS DOS treats the name, slash and switches as %0. However, DR DOS separates the name of the batch file into %0 and the parameters into %1.

Everywhere DOS sees a %1, it replaces that %1 with the value of the first replaceable parameter as entered on the command line.

DOS places no limit on the number of replaceable parameters you can enter on the command line, other than the 127-character limit DOS places on the command line.

DOS is flexible enough to allow you to use replaceable parameters as commands inside your batch files.

A.BAT performs a directory of the A drive. If a file specification is entered on the command line, it uses that as part of the Dir command.

RONNYMD.BAT first creates a subdirectory branching off the current subdirectory and then changes to that subdirectory. RONNYMD.BAT expects the user to enter a valid subdirectory name and has problems if no name is entered.

TOA.BAT tries to copy nine .DOC and .CMP files matching the nine replaceable parameters it expects to the A drive, as well as deleting the nine associated .BAK files. This batch file has significant problems.

11 Using more than ten replaceable parameters

As I mentioned in Chapter 10, DOS doesn't directly place a limit on the number of replaceable parameters you can specify on the command line. However, it does place an indirect limit. A command line is limited to 127 characters, and the name of the batch file and the first replaceable parameter as well as each set of replaceable parameters must be separated by a space, comma, or semicolon. If you run a batch file with a single character name, like A.BAT, and you give it only single-character replaceable parameters, then you will have room for 63 replaceable parameters. While 63 is the upper limit for replaceable parameters, most batch files use far less.

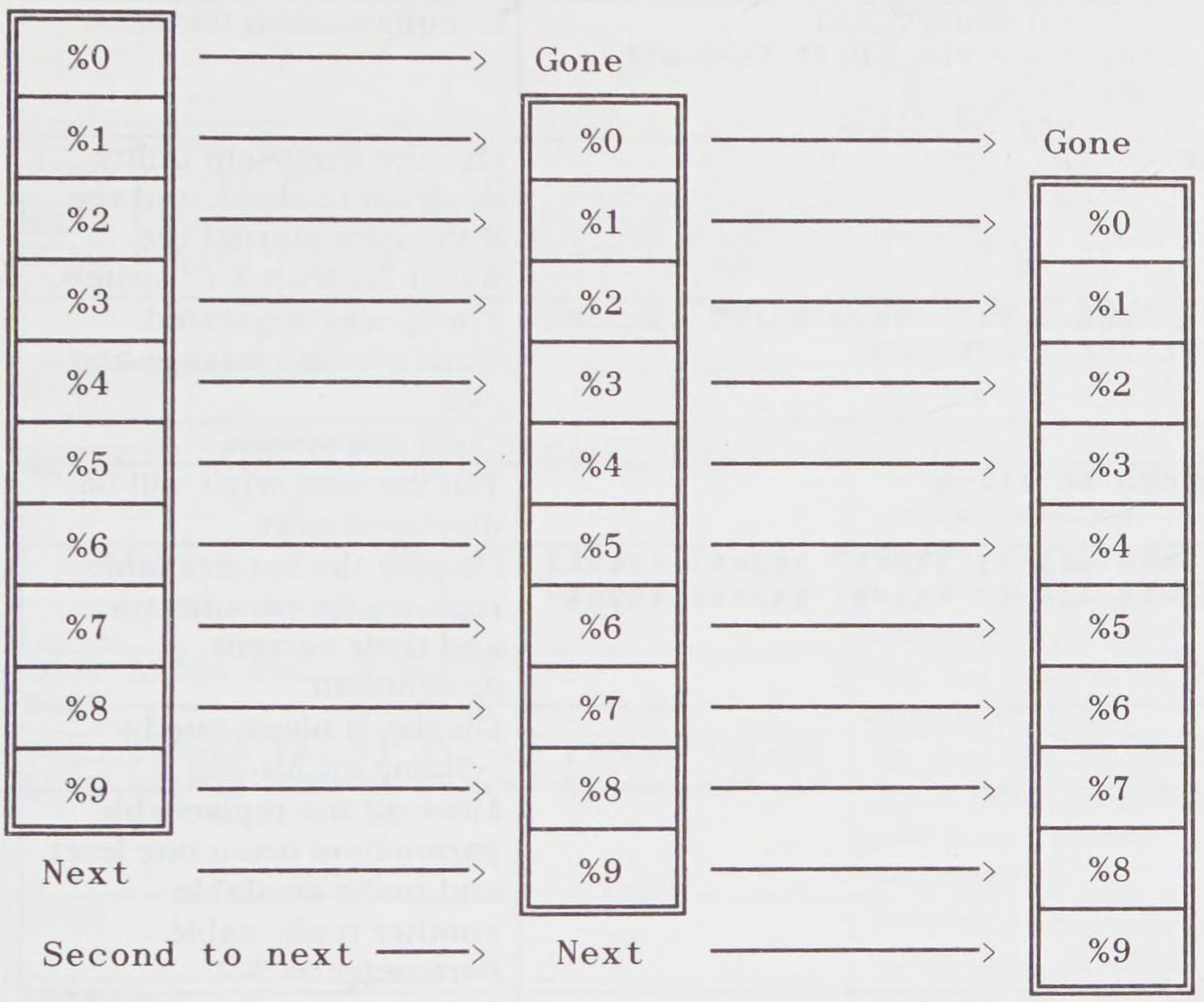
No matter how many replaceable parameters you pass to a batch file, only the replaceable parameters %0–%9 are valid in your batch file. While the particular set of ten replaceable parameters to which the batch file has access can be altered, a batch file is always limited to accessing only ten replaceable parameters.

☑ Remember: A batch file can only access ten replaceable parameters.

The SHIFT command

Because a DOS batch file can only "see" ten replaceable parameters, DOS offers the SHIFT command. The SHIFT command discards the %0 parameter, moves the remaining parameters down one value, and brings in a new %9 value if one exists. So, after a SHIFT, the value in %1 moves into %0, the value in %2 moves into %1, and so on. This is illustrated in FIG. 11-1.

You can better understand the SHIFT command with an example. Figure 11-2 shows SHOWSHIF.BAT, the batch file I will use to illustrate the SHIFT command.



11-1
The SHIFT command discards the %0 replaceable parameter, moves the remaining replaceable parameters down one level, and then makes available one more replaceable parameter in the %9 slot.

But before that, let's talk briefly about the table I've used to present SHOWSHIF.BAT.

Batch file tables

As you have seen in the first ten chapters, I've presented short batch files in the text. That is a neat way to show short batch files that need only a little explanation, and I will continue to use this approach where appropriate.

However, as we move through this book, many of our batch files will start to become longer and more complex. That is why I've developed the concept of this table. The batch file code is listed on the left side of the table. For each single line or group of very similar lines, the right side of the table has a detailed explanation of the function of that line or group of lines. That way, you can be assured that you can fully understand how each batch file works.

I've varied the width of the left side of the tables to avoid—to the extent possible—having to use word wrap on the batch file commands. When it proved to be necessary to use word wrap, I've indented every line except the first one. So, a batch file line in the table where word wrap was necessary might look like this:

ECHO This Is An Example Of A Batch Command Where Word Wrap Is Needed	Tell the user what is happening next.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: SHOWSHIFT.BAT`
`REM PURPOSE: Show The Shift Command`
`REM VERSION: 1.00`
`REM DATE: May 12, 1992`
	Documentation remarks.

`NEEDHELP %0 %1`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF ERRORLEVEL 1 ECHO Show SHIFT Command` `IF ERRORLEVEL 1 GOTO END`	If help was requested, show a brief message and exit.
`CLS`	Clear the screen.
`ECHO Before Shifting` `ECHO ----------------`	Tell the user what will be displayed next.
`ECHO %%0=%0 %%1=%1 %%2=%2 %%3=%3 %%4=%4` `ECHO %%5=%5 %%6=%6 %%7=%7 %%8=%8 %%9=%9`	Display the ten available replaceable parameters and their current assignment.
`ECHO`	Display a blank line by echoing an Alt-255.
`SHIFT`	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
`ECHO After First Shift` `ECHO -----------------`	Tell the user what will be displayed next.
`ECHO %%0=%0 %%1=%1 %%2=%2 %%3=%3 %%4=%4` `ECHO %%5=%5 %%6=%6 %%7=%7 %%8=%8 %%9=%9`	Display the ten available replaceable parameters and their current assignment.
`ECHO`	Display a blank line by echoing an Alt-255.
`SHIFT`	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
`ECHO After Second Shift` `ECHO ------------------`	Tell the user what will be displayed next.
`ECHO %%0=%0 %%1=%1 %%2=%2 %%3=%3 %%4=%4` `ECHO %%5=%5 %%6=%6 %%7=%7 %%8=%8 %%9=%9`	Display the ten available replaceable parameters and their current assignment.
`ECHO`	Display a blank line by echoing an Alt-255.
`PAUSE`	Pause the program until the user presses a key.
`CLS`	Clear the screen.

11-2
SHOWSHIFT.BAT shows the SHIFT command in operation.

11-2
Continued.

Batch File Line	Explanation
`SHIFT`	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
`ECHO After Third Shift` `ECHO ----------------`	Tell the user what will be displayed next.
`ECHO %%0=%0 %%1=%1 %%2=%2 %%3=%3 %%4=%4` `ECHO %%5=%5 %%6=%6 %%7=%7 %%8=%8 %%9=%9`	Display the ten available replaceable parameters and their current assignment.
`ECHO`	Display a blank line by echoing an Alt-255.
`SHIFT`	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
`ECHO After Fourth Shift` `ECHO ----------------`	Tell the user what will be displayed next.
`ECHO %%0=%0 %%1=%1 %%2=%2 %%3=%3 %%4=%4` `ECHO %%5=%5 %%6=%6 %%7=%7 %%8=%8 %%9=%9`	Display the ten available replaceable parameters and their current assignment.
`ECHO`	Display a blank line by echoing an Alt-255.
`SHIFT`	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
`ECHO After Fifth Shift` `ECHO ----------------`	Tell the user what will be displayed next.
`ECHO %%0=%0 %%1=%1 %%2=%2 %%3=%3 %%4=%4` `ECHO %%5=%5 %%6=%6 %%7=%7 %%8=%8 %%9=%9`	Display the ten available replaceable parameters and their current assignment.
`ECHO`	Display a blank line by echoing an Alt-255.
`GOTO END`	Jump to the end of the batch file.
`:END`	Label marking the end of the batch file.

I originated this method of displaying my batch files in the third edition of my *MS-DOS Batch File Programming* book, and I've used it in all my "batch" books since then. While harder to set up, this method has provided my readers with unparalleled details on the operation of the more complex batch files in my books.

Take a moment and run SHOWSHIF.BAT with a command like:

```
showshif 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

What SHIFT can do

As you can see, each time the batch file issues a SHIFT, the replaceable parameters move down one level with another replaceable parameter becoming available as %9.

The one thing you cannot see is what happens to the value contained in %0 after the SHIFT command. When SHOWSHIF.BAT is first run with the command line just shown, the value in %0 is "showshif." When the first SHIFT is issued, the "showshif" is removed to make room for moving the contents of %1 into %0. That value is unrecoverable. There is no "UNSHIFT" command. Once you have shifted a value away, there is no way to access it, so use SHIFT with care.

Warning: There is no "UNSHIFT" command so use the SHIFT command with care.

Using the SHIFT command

The SHIFT command has two major purposes. First, by moving the parameters down into a lower replaceable parameter, a single replaceable parameter can be used for all coding by forcing the program to loop through the code. This makes it much easier to update the code when necessary. (I describe looping starting in Chapter 12.) Second, it allows the batch file to handle more than ten replaceable parameters. Just keep in mind that the total number of replaceable parameters is constrained by the 127-character command line limitation and the requirement to separate each replaceable parameter on the command line.

Of the three sample batch files introduced in Chapter 10, only TOA.BAT can benefit from the SHIFT command. First, take a quick look back at the original version of TOA.BAT in Chapter 10. Now, compare that to the modified version shown here:

```
@ECHO OFF
REM NAME:    TOA.BAT
REM PURPOSE: Copy .CMP And .DOC Files To A Drive
REM          While Deleting .BAK Files
REM VERSION: 2.00
REM DATE:    May 12, 1992
DEL %1.BAK
COPY %1.DOC A:
COPY %1.CMP A:
SHIFT
REM The Last Four Lines Are Repeated Verbatim
REM Eight More Times. I've Removed Their Code
REM From The Figure For The Sake Of Space.
```

With this new version, the SHIFT command is used repeatedly so that TOA.BAT is always dealing with the same small section of code

```
DEL %1.BAK
COPY %1.DOC A:
COPY %1.CMP A:
SHIFT
```

which it uses nine times. If there was a way to reuse that code, TOA.BAT would be much shorter. As it turns out, there are two approaches to reusing this section of code: looping and subroutines. Looping is covered in Chapter 12, and subroutines are covered in Chapter 22. We will revisit TOA.BAT in both chapters.

Summary

DOS places no limit on the number of replaceable parameters you can enter on the command line, other than the 127-character limit DOS places on the command line.

The SHIFT command discards %0, moves %1 to %0, %2 to %1, and so on. It also brings another replaceable parameter into the %9 slot.

The SHIFT command is not reversible.

When combined with looping or subroutines, SHIFT allows one section of a batch file to work with all the replaceable parameters.

12 Looping and program flow

So far, our simple batch files have done what I recommend my students do with term papers—start out at the beginning and end at the ending. While that results in some very useful batch files, it doesn't give the batch file any decision-making abilities. The batch file must start at the top and progress downwards even if the conditions that exist at the time of execution are different than when the batch file was designed.

The process of designing our batch files to react differently to different circumstances will span several different chapters. We begin in this chapter by examining several ways to alter the normal top-down progression of a batch file. By doing this, we create several alternative paths that the batch file can take depending on circumstances.

Later, we will build on this concept by introducing decision-making tools to the batch file so it can choose from these different paths through the batch file depending on conditions. So, this chapter shows how to define multiple paths through your batch file, while a later chapter shows how to have the batch file pick a path.

Pseudocode

If you program in a general purpose computer language—like C or Pascal—you generally follow these steps:

1. The programs begin as an idea inside your mind.
2. You translate your idea into computer code. (Most computer tutorials recommend you draw a flowchart first, but no one ever does that!)
3. You run a "compiler" that converts the code that you wrote and understand into a program that the computer can understand.
4. Once your code is free of enough bugs for the compiler to successfully convert it into a program, you can begin running and testing it.

Pseudocode is really Step 1.5. Because no one ever does a flowchart anyway, pseudocode is a way of flowcharting without all the rigors.

Of course, with batch files or any language like GWBasic that acts as an interpreter, Step 3 disappears. Batch files don't need to be compiled because COMMAND.COM reads each line and interprets it as it executes rather than creating a program file of executable code.

Remember: Pseudocode is computer code written for a human to understand. ☑

Pseudocode is computer code written in English for later conversion into a programming language. Consider the pseudocode example of a loop:

Mark the top of a loop
Do something
If some test is met, jump out of this loop
Go to top of loop

This program goes around in a continual loop until a certain condition forces it to jump out of the loop. However, the real point of this example is that you understand it without knowing any of the associated batch commands!

In addition, pseudocode is also language-independent. If I were to flesh out this example with a few actions and specify the condition for when to jump out of the loop, a Basic or C programmer could write a program to perform the same task in their respective languages.

Pseudocode is not new. I originally learned pseudocode while learning to program in dBASE II on an Apple II with a CP/M card over ten years ago. I will be using pseudocode occasionally in this book to explain the flow of longer batch files because it makes the logic very easy to follow.

Naming program lines

The first thing our loop pseudocode requires is that we "mark the top of a loop." In a batch file, there's no specific marking for the boundary of a loop. Instead, you can give any line a name so the batch file can then jump to that line. To name a line, start with a colon and then type in a name with the following characteristics:

- It must follow the colon with no spaces between the colon and name. While not required by DOS, it is a good idea to left-justify the colon and name. Some batch file compilers won't work with a batch file unless the label is left-justified.
- The name should have eight characters or less. Actually, the name can be longer than eight characters; DOS just ignores everything after the eighth character. That can cause problems if you have labels like GOTOTHIS1 and GOTOTHIS2, because they're identical through the first eight characters. As a result, both GOTO GOTOTHIS1 and GOTO GOTOTHIS2 will cause DOS to jump to the label that appears first in the batch file.
- There can be nothing after the name.
- Capitalization doesn't matter. The labels :END and :End work the same.
- There can be no spaces in the name.
- While you don't have to, it's a good idea to make the name mean something, like top or end.

These rules are quickly summarized in TABLE 12-1.

Table 12-1
Line-naming rules.

1. Begin the line to name with a colon in the left-most position.
2. Immediately follow the colon with a line name of eight or fewer characters.
3. Do not enter anything after the name.
4. Capitalization does not matter.
5. Do not use spaces in the name.
6. Try to have the label name mean something.

Notice: While not a requirement of DOS, some batch file compilers demand that the label be left-justified. So, unless you have a good reason to do otherwise, you should left-justify your labels.

The next two lines in the pseudocode above were "do something" and "if some test is met, jump out of this loop." The "do something" line represents your typical commands in a batch file. For example, a segment of the TOA.BAT batch file we have examined over the last two chapters might look like this:

```
:MAINPART
DEL %1.BAK
COPY %1.DOC A:
COPY %1.CMP A:
SHIFT
```

The first line is the label, and the next four lines are the "something" this section is explaining. The "if some test is met, jump out of this loop" line represents logic testing. This topic is covered in detail in the next chapter.

The GOTO command

The last line in the pseudocode above is "go to top of loop." In a batch file, you jump to another section of the batch file by jumping to a line with a name. Because the line containing the name doesn't perform any action, the batch file behaves as though it jumps to the line following the name.

The command used to jump to another line is the GOTO command. The GOTO must be followed by the name of the label to jump to without the colon that precedes the label in the batch file itself.

GOTO has two specific uses: repeating a series of commands more than once (in other words, looping) and transferring control to another section of the batch file. We will look at both uses.

Looping

One important use of GOTO is to loop through a section of code more than once. We can see this with a couple of simple examples. We'll call the first one ENDLESS.BAT:

```
@ECHO OFF
REM NAME:    ENDLESS.BAT
REM PURPOSE: Demonstrate GOTO Command
REM VERSION: 1.00
REM DATE:    May 13, 1992
:TOP
ECHO Before A GOTO Command
```

```
GOTO TOP
ECHO After A GOTO Command
```

There is a copy of ENDLESS.BAT on the disk that comes with this book. If you run ENDLESS.BAT, you will see a series of "Before A GOTO Command" messages scrolling up the screen. You will have to press Control-Break to stop the ENDLESS.BAT.

Let's review the operation of this batch file and you'll see why. The steps in the batch file are as follows:

1. Process the `@ECHO OFF` command.
2. Skip over the REM commands.
3. Skip over the :TOP label. (Labels are completely ignored except when the batch file is looking at the batch file to figure out where to jump to after a GOTO command.)
4. Run the `ECHO Before A GOTO Command` line, displaying its message.
5. The `GOTO TOP` line is executed, causing the batch file to jump back up to Step 3.

Because it jumps back to the :TOP label each time it reaches the GOTO command, this batch file will continue looping until you force it to stop. This is called an endless loop because, if left alone, it would continue without ending.

By the way, you can stop ENDLESS.BAT by pressing Ctrl-Break or Ctrl-C and responding "yes" to the "Terminate batch job (Y/N)?" message. In the next chapter, we will see how to add logic testing to batch files so that the batch file itself can figure out if it needs to jump out of a loop rather than depending on the user pressing Ctrl-Break. This "stopping logic" is almost always needed when using the GOTO command to construct a loop.

The steps between the :TOP label and the GOTO command in ENDLESS.BAT form a multi-line loop. It is standard programming practice to indent everything inside a loop—that is, everything except the top (the label) and the bottom (usually the GOTO command). This helps you visualize the loop as you look at the code and actually forms documentation of sorts. DOS doesn't require that the commands be indented; in fact, the indentation slows down the batch file a little. It does, however, make the batch file easier for someone to understand when they look at it.

Using this information, we can modify the TOA.BAT batch file we've looked at over the past two chapters. Look back at the version in Chapter 11. In that version, it performed the same few steps nine times. The new version uses a loop to consolidate those few steps inside a single loop. It looks like this:

```
@ECHO OFF
REM NAME:    TOA.BAT
REM PURPOSE: Copy .CMP And .DOC Files To A Drive
REM          While Deleting .BAK Files
REM VERSION: 3.00
REM DATE:    May 12, 1992
:TOP
 DEL %1.BAK
 COPY %1.DOC A:
 COPY %1.CMP A:
 SHIFT
GOTO TOP
```

The first two versions of TOA.BAT had problems if fewer than nine replaceable parameters were entered because it was designed to handle exactly nine. This third version has a different problem. Because there is nothing in the batch file to cause it to stop, it wants to continue running forever. Of course, it soon runs out of replaceable parameters, and you must press Ctrl-Break to stop it. We will see how to avoid this in Chapter 16.

Transferring control

While it makes little sense, the following batch file is completely legal:

```
@ECHO OFF
GOTO END
ECHO This Message Never Seen
ECHO This Message Skipped Too
:END
```

When the batch file processes the second line, control is immediately transferred to the last line of the batch file and it terminates. It makes little sense because it has lines that are never executed.

While the unconditional jumping shown in this example makes little sense, it often makes good sense for a batch file to jump to different locations depending on conditions. Let's look as the pseudocode for the version of TOA.BAT we will be producing in a later chapter.

Top Of Loop
Test For Another Replaceable Parameter Pending
    If No Then Jump To End Of Batch File
    If Yes Then Continue
    *Erase The *.BAK File*
    Copy The .DOC File To A Drive
    Copy The .CMP File To A Drive
    Shift Replaceable Parameters Down One Level
    Go Through Loop Again

We haven't yet covered logic testing so we cannot yet produce this batch file as written, but you can see how the ability to jump out of that loop when the batch file "runs out of" replaceable parameters is useful.

In general, we will use this unconditional transfer of control to jump to different sections of the batch file, depending on the conditions the batch file finds when it begins operation. Of course, that means you must write different sections of the batch file to deal with these different conditions.

Quick exit

Many of the batch files in this book will have one or more `GOTO END` commands to exit the batch file quickly without processing the remainder of the batch file. When this command is executed to terminate a batch file and not a batch file functioning as a subroutine, there is actually a quicker way. In your batch file subdirectory, enter the command:

```
TYPE NOFILE > QUIT.BAT
```

where *NOFILE* is the name of a nonexistent file.

As an aside, make sure that you don't already have a QUIT.BAT because this command would erase it. This line creates a 0-length file called QUIT.BAT. Now,

replace the GOTO END commands with the QUIT command. Running QUIT.BAT without CALL keeps control from returning to the original batch file; and because QUIT.BAT has no commands, the batch file immediately terminates.

With some earlier versions of DOS, you cannot use this method to exit a subroutine when you want control to return to the batch file that called the subroutine because the subroutine terminates and its calling program does not retain control. This was corrected in DOS 5. Also note that DOS will not copy a 0-length file, so you will have to recreate QUIT.BAT on every machine you want to use it on. That is why I did not include a copy on the disk that comes with this book.

Warning: This new QUIT command shouldn't be used in a batch file that's running as a subroutine unless you have upgraded to DOS 5 or DOS 6.

Handy formatting option

When we mentioned looping earlier, I said that you should indent everything between the top and bottom of the loop as a visual aid in debugging and maintaining the batch file. If you do that, the loop from TOA.BAT looks like this:

```
:TOP
 DEL %1.BAK
 COPY %1.DOC A:
 COPY %1.CMP A:
 SHIFT
GOTO TOP
```

As we begin developing longer batch files, you will find these to be very compartmentalized. In many cases, the batch files will have many different sections for handling specific tasks. For example, a "full-blown" version of TOA.BAT might have sections to perform the following:

- Display help when the user starts the batch file with a /? switch.
- Display an error message when the user fails to enter a replaceable parameter.
- Display an error message when the user specifies a filename as a replaceable parameter and the .DOC version of that file doesn't exist. (Not all files have associated .CMP and .BAK files, so don't give an error message if one of those is missing.)
- Display an error message when the target disk is full or if other copying problems occur.
- Perform the actual copying.

Clearly, you need a method to keep these sections separate, both in execution and visually.

Code separation

Making sure the batch file executes only the right section of code is fairly easy. You jump into the appropriate section of code using the GOTO command as the result of a logic testing statement. At the end of the section, the batch file jumps to the next section of the batch file. You can see the TOA.BAT segment reproduced earlier. It doesn't need to jump into the loop because it reaches that sequentially, but it jumps at the end of the loop.

Visual separation

One way to visually separate the sections is to place one or more blank lines between the different sections. If the section is a loop, then you have the visual clue of the indenting used inside the loop. I tend to use the same form of indenting with

sections that are not loops. For example, a section of TOA.BAT to display an error message when the user fails to enter a replaceable parameter might look like this:

```
:NOTHERE
 ECHO You Started TOA.BAT
 ECHO Without A Replaceable Parameter
 ECHO This Information Is Required For TOA.BAT To Run
 ECHO When You Start TOA.BAT
 ECHO List The Names Of The Files
 ECHO To Copy On The Command Line After The TOA Command
GOTO END
```

Now, you have a visual clue that this section of code is a integral section of code.

The FOR command

Once we cover logic testing in the next chapter, we'll have everything that you'll need to know to write complex loops using the GOTO command. However, one limitation of looping via the GOTO command is that you must provide the parameters of the loop when you code the batch file. For example, you cannot go through the loop once for each .DOC file or for every file on the B drive. For that, you need the FOR command.

Looping with the FOR command

In its simplest form, the FOR command causes DOS to loop through a series of files and perform a single action on those files. There are two forms of the command. The first takes a list of files:

```
FOR %%h IN (CHAPTER1.BAK CHAPTER2.BAK) DO ERASE %%h
```

The second form uses DOS wildcards to calculate all applicable files:

```
FOR %%j IN (*.BAK) DO DEL %%j
```

In general, the command is

```
FOR %%variable IN (set) DO Command
```

'The FOR command must be on one line. Also, FOR commands cannot be nested, so only one FOR command can be on each line. The %%variable must be a single character: for example, %%A through %%Z. In addition, if you reference the %%variable after the DO command, you must use the same capitalization as you used in the beginning. For example,

```
FOR %%J IN (*.*) DO ECHO %%J
```

and

```
FOR %%j IN (*.*) DO ECHO %%j
```

are valid because both %%J's are the same case, but the command

```
FOR %%J IN (*.*) DO ECHO %%j
```

is invalid because one J is uppercase and the other is lowercase.

☑ Remember: The FOR command must be all on one line and cannot be nested.

DOS will let you use %0 through %9 as variables in the FOR command, but it is a bad programming habit because DOS uses these as replaceable parameters. You can also use the FOR command from the command line prompt without a batch

file. The only change is that you only use one percent sign rather than two, so the %%variable must be %variable.

Warning: DOS allows you to use %0–%9 as the variable in a FOR command. Don't do it. This is poor programming because DOS uses %0–%9 as replaceable parameters.

Local variable

When we worked with replaceable parameters back in Chapter 10, we saw that the replaceable parameters were global variables. That is, you could use %1 anywhere in your batch file and the command would have access to the value stored in %1. However, the variables created by the FOR command are local variables. Their value is only usable while the batch file is running. That is, the following batch file segment

```
@ECHO OFF
FOR %%J in (1 2 3) DO ECHO %%J
ECHO %%J
```

would not have a value for %%J to echo when it reached the third line so it would simply echo "%J." (Recall that you use two percent signs in an ECHO command to echo a single percent sign.)

Remember: FOR variables are local variables whose value is only usable within the FOR loop.

Running commands with a FOR loop

Every so often, the computer magazine hint columns will have a hint from someone who has just discovered that the things you put in the list of variables for a FOR command can be commands themselves. The authors of these hints are excited because the batch file runs faster this way; if you remember, DOS processes batch files one line at a time, and this strategy puts more commands on a single line.

Don't be fooled. While the batch files are truly a little faster, this method has a couple of drawbacks that greatly outweigh this slight dose of speed, with the biggest drawback being human readability. You and I could probably look at a batch file with the lines

```
DIR
CHKDSK
CD\
```

and tell exactly what it is supposed to do. However, the function of a batch file with the line

```
FOR %%j IN (DIR CHKDSK CD\) DO %%j
```

isn't nearly so clear. In the end, you will waste more time writing and debugging such complex code than you will save by stacking more commands on a single line.

This method has another drawback: you cannot use this trick with any command that requires more than one word. For example, you might want to run CHKDSK and then beep the bell by echoing a ^G. With two lines, you would enter the commands

```
CHKDSK
ECHO ^G
```

Using the FOR command, you would enter the command

```
FOR %%j IN (CHKDSK ECHO ^G) DO %%j
```

but because of the space, DOS would actually try to run

```
CHKDSK
ECHO
^G
```

There is yet another problem. When the slash command is used in the FOR command, it is treated as a divider. So the command

```
FOR %J in (DIR/W CHKDSK/F FORMAT/S) DO %J
```

has six different elements (DIR, W, CHKDSK, F, FORMAT, and S) in its set.

For a quick reference chart for the FOR command, look at TABLE 12-2.

Table 12-2
FOR command summary.

1. The syntax of the command is
   `FOR %%J IN (Set) DO` *`Command`*
2. The (Set) can contain a list of files, wildcards, or simply a list of items to process.
3. The command to execute might or might not use the %%J variable.
4. FOR commands can also be issued from the command line; just replace the %%J variable with a %J variable having only a single percent sign.
5. If the %%J variable is referenced by the command, that reference must use the same capitalization as was used at the beginning of the FOR command.
6. The FOR command cannot be nested.
7. The variables %0–%9 should not be used as FOR command variables although DR DOS allows it.
8. The %%J variable is a local variable that no longer contains a value once the FOR command terminates.
9. Command can be used in the (Set) list and executed by the Command; however it is poor programming.
10. Commands requiring a command line switch that is preceded by a slash cannot be included in the (Set) when executing commands using the FOR command.

Watching for wildcards in replaceable parameters

The FOR command loops through all the elements in its set and performs its command once per element. When an element in the set contains wildcards, the FOR command first expands the wildcards. For example, the command:

```
FOR %%J IN (*.*) DO DEL %%J
```

in a batch file will delete every file in the current subdirectory without triggering the DOS "All files in directory will be deleted! Are you sure (Y/N)?" error message!

The batch file TESTWILD.BAT, shown below, illustrates how to avoid this problem:

```
@ECHO OFF
IF (%1)==() ECHO Requires Parameter To Work
IF (%1)==() GOTO END
```

```
IF NOT EXIST %1 ECHO File Does Not Exist
IF NOT EXIST %1 GOTO END
FOR %%J IN (%1) DO SET FROMFOR=%%J
IF %1==%FROMFOR% ECHO %1 Does Not Contain Wildcards
IF NOT %1==%FROMFOR% ECHO %1 Contains Wildcards
:END
```

First, it uses an `IF (%1)==()` test to make sure a replaceable parameter was entered. If so, it goes to the next stage where it uses an `IF NOT EXIST %1` test to exit if no files exist matching the replaceable parameter. If at least one file matches the replaceable parameter, it loops through all of them using a FOR command and sets an environmental variable equal to the expanded value. If no wildcards were used, the name of the file and the name of the expended file from the FOR command are identical and the `IF %1==%FROMFOR%` test passes. If the FOR command had to expand wildcards, then the name of the file and the replaceable parameter will not match and this test will fail.

This gives you a way to test to see if a replaceable parameter contains wildcards when that is necessary. This is especially useful when creating directories or deleting files when a wildcard could wreak havoc.

FOR command quirks

The FOR command has two quirks, both of which can be useful under certain circumstances. The first quirk is how it processes elements of its set when it is preceded with a forward slash (/). The variable is split in two with the first element being the first character and the second element being the remainder. For example, if you enter this command from the command line:

```
FOR %J IN (/ABC /XYZ /123) DO ECHO %J
```

you might expect to see three messages, "/ABC," "/XYZ" and "/123" but what you actually see are six, "A," "BC," "X," "YZ," "1" and "23." Not only is the first character separated, the forward slash is stripped off as well.

The second quirk involves what the FOR command will actually run after the DO component. Only internal commands and programs are allowed after the DO component—no batch files are allowed. You can put a batch file there but the FOR loop will abort after processing the first element in its set. This makes things difficult if you want to use the FOR command to do several steps, like copying a file to a backup floppy disk and then deleting it. Fortunately, a slight trick lets us run a batch file in a FOR loop. In a batch file, the CALL command is used to run another batch file. Well, the CALL command is an internal command so the command:

```
FOR %%J IN (%1) DO CALL BATCH %%J
```

would loop through all the files matching the first replaceable parameter and run BATCH.BAT for each one, passing it the file name as the first replaceable parameter.

This second quirk gives us a way to avoid one of the major limitations of the FOR command—its inability to run more than one command for each element. By using a batch file, as many commands can be executed for each element as are required. The one thing to notice is that the element must be passed to the batch file in the FOR command because FOR variables are local variables and are not available in the batch file. The last command illustrates this.

Summary

Pseudocode is the process of outlining programs in English without worrying about which language will be used to create the program. Pseudocode is later turned into a program.

Lines are given a name in a batch file by preceding an eight-character or shorter name with a colon.

The GOTO command causes control to jump from the current line to the line named as part of GOTO.

The GOTO command can be used to loop or to permanently transfer control to another section of the batch file.

Loops created with GOTO need some logic added to tell the batch file when to stop looping.

Once the 0-length file QUIT.BAT has been created in a subdirectory in your path, the `QUIT` command can be used to quickly terminate a batch file without the need for a `GOTO END` command and the corresponding :END label.

When creating loops with the GOTO command, it is useful to indent everything between the label and the GOTO command to visually represent the loop.

The FOR command constructs a single-line loop that is automatically executed once for each item in the (set) of items specified with the command.

The FOR command must be all on one line and cannot be nested.

Using the code shown in TESTWILD.BAT, you can test replaceable parameters to see if they contain wildcards before using them in a FOR loop if that is required.

Due to a quirk in the FOR command, when an element in its set is preceded with a forward slash, that element is split into two components, one containing the first character and the second containing the remaining characters. The forward slash is dropped.

In order to run a batch file via a FOR loop, the name of the batch file must be preceded with a CALL command.

13 The environment

DOS sets aside a portion of memory for storing system-wide variables. This area is called the *environment*, and the variables stored there are called *environmental variables*. In addition to storing the value of environmental variables, this area stores the prompt, path, and COMSPEC. Technically, these three values are nothing more than environmental variables, although they have such specific functions that they are usually treated differently. Because they are so important, let's take a moment to look at each one.

The prompt

The default prompt is C>, which tells you almost nothing (except that C is the default drive). You can use the PROMPT command to change the prompt to a wide range of prompts. The PROMPT command is normally just used in the AUTOEXEC.BAT file; when used by itself, it resets the prompt to C>.

Any printable character string can be included in the PROMPT command. In fact, one of the first tricks most computer users learn is to include their name or company name in the prompt. Special characters can be included in the prompt with the commands in TABLE 13-1. The most popular prompt is

```
SET PROMPT=$p$g
```

This command adds the current subdirectory to the default disk display. It's important to remember that any prompt you develop is stored in the environmental space, along with the path and environmental variables. A long prompt combined with a long path, and an environmental variable might require you to expand your environmental space, as explained later in this chapter.

Table 13-1
Metacharacters to use in the PROMPT command.

Command	Action
$$	Display a dollar sign.
$_	Include a carriage return and line feed.
$b	Display a vertical bar.
$d	Display the date.
$e	Include an escape. This is useful for sending Ansi escape sequences via the prompt command.
$g	Display a greater-than sign.
$h	Display a backspace—thus deleting the prior character.
$l	Display a less-than sign.
$n	Display the current drive.
$p	Display the current subdirectory.
$q	Display an equal sign.
$t	Display the time.
$v	Display the DOS version.

The path

DOS will accept four types of commands: internal commands, DOSKEY macros .EXE and .COM programs, and batch files (.BAT). Every time DOS receives a command, it first checks to see if the command is a DOSKEY macro; if it is, it runs it. If it's not a DOSKEY macro, it next checks to see if the command is an internal command, like ERASE; if so, it executes it. If the command isn't an internal command, DOS next checks the current subdirectory for an .EXE program by that name, then a .COM program, and finally a .BAT file. If DOS finds a program with the correct name, it executes that program. If DOS doesn't file in the current directory, it searches the path for a .COM, .EXE, or .BAT file. This is explained in detail in Chapter 2.

Its importance here is that the path is stored in the environment. The normal syntax for creating environmental variables will work, so the command

```
SET PATH=C:\;C:\DOS;C:\WINDOWS
```

will work just fine. A shortcut is also available. The command

```
PATH=C:\;C:\DOS;C:\WINDOWS
```

will also create a path.

This second approach to creating a path has two advantages over the first. First, paths tend to grow larger and larger but the path command—like all DOS commands—is limited to 127 characters. Removing the "SET "reduces the length of the command by four and thus allows your path to contain four additional characters. Second, paths created with the PATH= command are converted to all uppercase, while those created with SET PATH= are not. We will find this useful later when converting characters to all uppercase for easy comparison.

I should mention that "technically" the path is not limited to 122 characters (127, less the five needed for PATH=); rather, the DOS command line is limited to 127

characters and *that* forces this limitation. There are utilities available that will write a longer path directly to the environment without needing a longer command line, and thus give you a path that exceeds 122 characters. However, every programmer "knows" the path will be 122 characters or less, so many programs do not allow for a longer path and will end up ignoring any portion of the path beyond the generally expected 122 characters. Of course, any other environmental variable could be expanded in the same fashion. However, there is no need for any other environmental variable to become this long, so—as far as I know—no one has written a program to allow other environmental variables to become longer than DOS allows.

COMSPEC

COMSPEC is short for COMmand SPECification. It tells DOS where to find COMMAND.COM when it's overwritten by a program.

DOS takes up a lot of memory. When your computer is short of memory, things can get crowded. The lack of memory might prevent some programs from being run or even limit the size of others. DOS solves this problem by making part of itself provisionally resident (or transient) in memory. You need this part to enter DOS commands but not to run the application programs. If a program needs this space, it can overwrite the transient portion of DOS.

When you exit an application program that's overwritten the transient portion of DOS, part of DOS is missing. Thus, if you were to use this version, you wouldn't be able to enter most internal commands. DOS replenishes itself by rereading portions of COMMAND.COM into memory.

Note: Because DOS needs to occasionally reread COMMAND.COM, it isn't a hidden file like the other two system files.

Usually, DOS reloads itself from the drive it booted from. Using

```
SET COMSPEC=C:\COMMAND.COM
```

you can force DOS to reload itself from some other place. You can also change COMSPEC using the SHELL command in the CONFIG.SYS file. Loading COMMAND.COM with the SHELL command rather than having the operating system load it automatically doesn't use any extra memory unless you expand the environment. And then, the only additional memory use is that of the extra environment. Many RAM-disk users copy COMMAND.COM to their RAM disk and then use the SET COMSPEC command to reload COMMAND.COM from the RAM disk. This is noticeably faster than reloading from disk.

Think of the environment as a scratch pad, where the operating system leaves notes for itself. As we will see, batch files can also use this scratch pad.

Environmental variables

Most programs distinguish between different types of variables. For example, you generally don't use the same type of variable for a string of characters as you do for numbers. The environment doesn't make this distinction. Every environmental variable is treated as a string of characters—even if it's all numbers.

Unlike filenames, environmental variable names aren't limited to a few characters. Therefore, it's a good idea to use descriptive names like SourceDrive and TargetDrive. Of course, these longer names take up more space in the environment; and, as we will see, environmental variable names are converted to uppercase. If an

environmental variable is used temporarily by a batch file, it is also a good idea for that batch file to delete that environmental variable once it no longer needs it so it is not using environmental memory.

☑ Remember: The environment is a scratch pad for the operating system.

Some notes on names

Many of the batch files in this book use environmental variables and almost every one uses them without concern for the name. That is, they go ahead and use a name without first checking to see if the name is already in use. That's ok for demonstration batch files but you need to be more careful on your system. If a batch file depends on an environmental variable existing when it runs, then you don't want another batch file altering its contents or even wiping it out!

Because environmental variable names can be long, you can avoid this problem by starting important environmental variable names with the name of the batch file that creates them. So if PASSWORD.BAT created an environmental variable called ALLOWED, you might name the environmental variable PASSWORD_ALLOWED. Because every batch file would use its unique name for the first portion of environmental variable names, they would all be unique.

You can also avoid problems by deciding on some names—like JUNK or TEMP—that you will never use for permanent information. That way, they can always be overwritten without concern for their contents.

While you do not want to accidentally overwrite a permanent environmental variable, you also want to avoid filling up your environment with environmental variables you no longer need. It's important that batch files clean up after themselves and delete any environmental variables that will not be needed later. This will go a long way in keeping you from running out of environmental space.

Increasing the size of the environment

Most compilers use one or several environmental variables to point to their libraries. In fact, a couple of them use the same variables to point to different libraries! I once got a frantic call from a developer using two compilers from the same company—Basic and C. Bóth compilers used the same environmental variable to point to their (different) libraries. He didn't know how to deal with this problem. I told him to construct two batch files, one to start each compiler. He could then have each batch file custom-set the variables for that compiler.

He called me back, even more frantic, because it wouldn't work. I had him type in SET at the DOS prompt and read the contents back to me. It turned out that he hadn't expanded the environment; and, with his long path and prompt, there just wasn't enough room to store all the information he was trying to shove into the environment. I walked him through the procedure to expand the environment, and then everything worked properly.

The default size of the environment for both DOS 5 and 6 is 256 bytes (which means that it can store 256 characters). For earlier versions, it was as low as 160 bytes. This storage space must store the COMSPEC value, the PATH, the PROMPT, and any variables you want to enter. If this isn't enough room, you will get an "Environment Full" error message. To correct the problem, you need to expand the environment.

Under DOS 5 and 6, expanding the environment is easy. Adding the following line as the first line of your CONFIG.SYS file will expand the environment:

```
SHELL=C:\COMMAND.COM /E:yyyyy /P
```

If you're running off floppy disks, change the C:\ to an A:\. The yyyyy can be a number from 512 to 32,768 (which allows an environmental space up to 32K!). The /P is required to force COMMAND.COM to automatically run the AUTOEXEC.BAT file. Because the Shell command is in the CONFIG.SYS file and the CONFIG.SYS file is processed prior to the AUTOEXEC.BAT file, the AUTOEXEC.BAT file will be bypassed without the /P switch. It is also interesting to note that the default size for the environment is 256 bytes, but the minimum you can specify with the /E switch is 512 bytes.

Expanding the environment with older versions of DOS can be tricker. Prior to version 3.2, you had to "patch" COMMAND.COM to expand the environment, a process that is fairly difficult if you are not experienced in such matters. That process is not explained in this book due to the very few number of readers using that old of a version of DOS. If you need the information, drop me a line in care of TAB Books at the address shown in the front of the book, and I'll mail it to you.

DOS 3.2 through 4.0 use a command similar to the previous one, but the values for the number after the /E: are calculated differently. The number you use represents the number of 16-byte segments to use for the environment. So a 20 gives you a 320-byte environmental space. The maximum value is 992 bytes, which is an xx of 62.

The previous SHELL statement uses the /E and /P switches. These aren't SHELL switches but rather COMMAND.COM switches. All the switches are as follows:

/?	This switch displays help about COMMAND.COM.
/C	This tells COMMAND.COM to run the command listed after the /C switch.
/E	This switch changes the environment sizes.
/K	Runs the program or batch file specified after the switch and then displays the DOS prompt.
/MSG	Stores error messages in memory as opposed to disk.
/P	This tells COMMAND.COM to load in permanent mode, which causes COMMAND.COM to set a couple of switches internally. One switch keeps it from unloading when you issue an Exit command, while another causes it to run the AUTOEXEC.BAT file after loading.

Note that you're required to put the .COM extension on the end of COMMAND.COM. And remember: these changes won't take effect until you reboot.

Additionally, make sure you have a bootable system disk in case you make a mistake because some mistakes will hang the computer. Because the CONFIG.SYS is processed before COMMAND.COM is loaded, you cannot press Ctrl-Break to stop the CONFIG.SYS if you make a mistake that causes the computer to lock up. When that happens, you must boot from a floppy disk to correct the problem. As another precaution, you should make a copy of your CONFIG.SYS file before trying this change.

While you should still have a bootable system disk handy for other general emergencies, DOS 6 avoids this problem. If you press F5 or both shift keys while it

displays the "Starting MS-DOS..." message, DOS will bypass your CONFIG.SYS and AUTOEXEC.BAT file this one time and give you a minimally configured system. (If you've selected disk compression, it will automatically process enough of your CONFIG.SYS file so you still have access to the compressed drives.) You can also press F8, instead of F5, and it will ask you about each line in the CONFIG.SYS file—giving you a way to bypass the offending line.

Deciding on a size

The best way I know of to decide on your final environmental size is to first make your environment very large, create all the environmental variables you need, find out how much space they need and then trim your environment back near that level—leaving a small cushion for the future.

COUNTENV.BAT can help you do that. Once you've created all your environmental variables, running COUNTENV.BAT will tell you the approximate amount of space your environmental variables use. It does this by piping your environment to a file so that the size of the file is approximately the size of your environmental variables. It then erases the file. The code from COUNTENV.BAT that actually performs the work follows:

```
SET > COUNTENV.@@@
ECHO Size Of COUNTENV.@@@ Is Approximate
ECHO Size Of Environment
DIR COUNTENV.@@@
DEL COUNTENV.@@@
```

The size of the file is only approximately the space required for the environmental variables for two reasons: in the file each environmental variable has two bytes after it (a return and line feed), while in memory it has only one (an ASCII 0); and the file has nothing after it, but in memory the environment has two ASCII 0's at the end. If you want the exact size of your environmental variables, add two to the size of the file and subtract the number of environmental variables you have from the resulting file. My environment has twelve variables and the file required 468 bytes, so my environmental variables require 468+2–12 or 458 bytes.

Environmental variables are transient

As mentioned earlier, all systems have a COMSPEC environmental variable, and most have a Path and Prompt environmental variable. Some hardware and software also creates environmental variables when you install them. Windows wants an environmental variable called Temp to point toward a place for its temporary files. My tape drive creates two, and my CD ROM creates three. You will find these added to your AUTOEXEC.BAT file. Usually, the installation program adds them automatically.

Variables in the environment are lost when you turn off or reboot the computer. Although these variables are transient, they can be very useful for communicating between different batch files or for storing information for use by more than one batch file.

☑ Remember: Variables in the environment are lost when you turn off or reboot the computer.

Putting variables into the environment

You can place environmental variables into the environment either with a batch file or interactively from the command line prompt. Either way, the syntax to place an environmental variable into the environment is

```
SET variable=value
```

Note that the only space is between SET command and the variable name. Don't use spaces around the equal sign.

If you include spaces around the equal sign, the name of your variable and its contents will be corrupted. With the command

```
SET RONNY = Test
```

you end up with a variable named "RONNY " containing the value " Test" and neither of these is what you want.

Warning: When using the SET command, be sure not to use a space before or after the equal sign.

Once you store a variable in the environment, you cannot edit its value. However, you can replace the contents of an environmental variable using the same command—thus overwriting the contents of the environmental variable with a new value. You can also delete an environmental variable with the command

```
SET variable=
```

where nothing follows the equal sign. Finally, you can view the contents of the environment by issuing the SET command by itself from the command line or in a batch file.

If you have DOS 5 or later, the DIR command allows you to store your preferences as an environmental variable called DIRCMD. For example, the command `DIR/OS/P` will list your files in order of size and pause after each screen of information. Add the command `SET DIRCMD=/OS/P` to your AUTOEXEC.BAT file, and the DIR command by itself will work the same as if you had added the "/OS/P" switches.

Using the environment in batch files

You can use a similar approach to transferring data between environmental variables. For example, if the environmental variable BEFORE contains a value you want to store in the environmental variable AFTER, the command to do this in a batch file is:

```
SET AFTER=%BEFORE%
```

Because of the percent signs, DOS replaces the %BEFORE% environmental variable with its contents; and because of the SET command, it then stores this value to AFTER. Two notes are important. First, the BEFORE environmental variable is surrounded by percent signs because we want to use its value, but the AFTER environmental variable isn't because we are using that name. Second, this command doesn't change the contents of the BEFORE environmental variable, so now we have the same information stored under two names. Of course, we could go ahead and work with either environmental variable to alter their contents.

Remember: Batch files can do more with environmental variables than you can from the command line.

What you can do with environmental variables

Programming languages generally let you perform mathematical operations on variables, but DOS doesn't have a command to do this on an environmental variable. However, the utility program Batcmd lets you add and subtract one from an environmental variable named MATH. Batcmd is included on the disk that comes with this book, and this feature is reviewed in Chapter 25.

Environmental variables can be used for the following:

- **Display.** A batch file can display the contents of any environmental variable by surrounding its name with percent signs. For example, `ECHO %PATH%` will display the path. Also, when the expanded line is longer than 127 characters, as it might be with a long path and the command `ECHO My Path Is: %PATH%`, then the entire message will not display and the computer might lock up.
- **Combining strings.** Two or more environmental variables can be displayed together. The technique is the same: just surround the respective names with percent signs. For example, if there are environment variables named Drive and File, the command `ECHO %DRIVE%%FILE%` will display them both side-by-side.
- **In place of text.** Environmental variables can be used any place where text would be used. For example, the command `PATH=%PATH%;C:\123` would append C:\123 onto the current path because the %PATH% is used just like the batch file entering the current path, and the command `COPY C:\AUTOEXEC.BAT %DRIVE%%FILE%` would copy your AUTOEXEC.BAT to the drive and filename stored under these environmental variables.
- **In place of commands.** A batch file can run a command stored to an environmental variable by having that environmental variable, surrounded by percent signs, on a line by itself.

Note, while all these techniques work from within a batch file, none of them work from the command line.

☑ Remember: Batch files can use environmental variables much more extensively than they can be used from the command line.

Using the environment when shelling out from a program

While batch files work with the original—or master—copy of the environment, every time you start a program, DOS passes that program a full copy of the environment. The program is free to alter its copy of the environment, but when the program terminates, its copy of the environment is also terminated without passing the changes back to the master copy of the environment.

Remember our comparing the environment to a scratch pad? Well, when DOS runs a program, it provides that program with a "photocopy" of its scratch pad in case the program needs any of the information it contains. However, before passing a copy of the environment to a program, DOS will trim off as much empty space as possible. It would trim all the free space except that—like all memory blocks—the environment is created and maintained in 16-byte (or one-paragraph) blocks. This trimming will therefore result in 0–15 bytes of free environmental space. When the program is done, DOS ignores any notes the program has made on its photocopy and just tosses the copy into the memory recycling bin.

The action of the environment while a program is running leads to an interesting problem and a useful shortcut for experimenting with the environment. First, the problem, and then the shortcut.

Environment problem when shelling out

Imagine you are setting up Windows to run a non-Windows program by running a batch file. For this example, we'll use WP.BAT from Chapter 10. This batch file will require one modification. Instead of running the menu as the last step, it will need to issue an EXIT command to unload the copy of COMMAND.COM Windows used to run the batch file. The modified batch file segment looks like this:

```
@ECHO OFF
C:
CD\WORD
WORD %1
CD\
EXIT
```

To avoid the problem of the user shelling out of Word and trying to run yet another program, you might want to add the modification examined in Chapter 3 of changing the prompt before starting Word and changing it back after Word terminates. That version of the batch file segment is shown below:

```
@ECHO OFF
C:
CD\WORD
PROMPT Type EXIT To Return To Word $P$G
WORD %1
PROMPT $P$G
CD\
EXIT
```

However, this batch file won't run as expected under Windows! The problem is that Windows discards the free memory from its copy of the environment so there's no more than seven bytes of free environmental space. When Windows loads a second copy of COMMAND.COM, that COMMAND.COM gets a copy of this reduced-size environment. Because the new "Type EXIT To Return To Word PG" prompt is much longer than the existing prompt, it will not fit and you'll get the "Environment full" error message.

This problem is not unique to Windows. If you shell out of Lotus or Word and try to run a batch file that adds information to the environment, you will find it also fails to run properly: Both Lotus and Word discard free environmental memory to have more conventional memory to run with and then pass along this reduced-size environment to COMMAND.COM when you shell out.

Increasing the size of the master copy of the environment won't help. This will just give the programs more free environment space to discard when they start.

The solution, then, is to find another approach to making environmental space available under programs that discard free environmental space. The steps are as follows:

1. In your AUTOEXEC.BAT file, create a long environmental variable containing enough information that, if deleted, would free up enough environmental space to make the environment usable under other programs. For this example, we'll call this variable JUNK. So the AUTOEXEC.BAT file would have a statement like

   ```
   SET JUNK=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

2. Before adding information to the environment in a batch file designed to run while shelled out of another program, delete the JUNK environmental variable. The modified version of the batch file segment is shown next:

```
@ECHO OFF
C:
CD\WORD
SET JUNK=
PROMPT Type EXIT To Return To Word $P$G
WORD %1
PROMPT $P$G
CD\
EXIT
```

3. If you are modifying a batch file that will be used both from the command line and from within a program, then deleting the JUNK environmental variable before adding information to the environment is OK. However, the current version of WP.BAT never recreates the JUNK environmental variable. As we said, when a program terminates, its copy of the environment is also terminated. Most programs shell out by running COMMAND.COM; this is always the case if you have to type EXIT to return to the original program. When that copy of COMMAND.COM terminates, changes to its environment are lost.

   However, when a dual-mode batch file is running from the command line, the environment isn't restored and the contents of the JUNK environmental variable won't be available for other batch files to delete. To avoid that, add a command at the end of any dual-mode batch files to recreate the JUNK environmental variable. That way, when they run by themselves, the environmental variable is recreated properly. When run from inside a program, there might not be enough free memory to recreate the variable, but in that case it doesn't matter.

☑ Remember: To create free environmental space while shelling out of another program, create a long environmental variable from the command line or in the AUTOEXEC.BAT file and then delete that variable to create free environmental space when shelling out of a program.

By following these three steps, you will always have free environmental memory no matter if your batch file is running from DOS directly or running while shelled out of Windows or any other program.

Environment benefits when shelling out

As explained earlier, after you shell out of a program, the changes you make to the environment are lost when you enter the EXIT command to return to the original program. However, as long as you follow the procedures just described to create a dummy environmental variable from your AUTOEXEC.BAT file that you can later delete, you will have free environmental space while shelled out of a program.

All that makes shelling out of a program the perfect place for testing batch files that modify the environment—especially if your system depends on a well crafted environment. No matter how many alterations you make to the environment, a simple EXIT command restores the environment to its pristine form.

When I'm writing a batch file that works with the environment, I usually use Microsoft Word rather than Editor to write the batch file. That way, when I'm ready to test the batch file, I can shell out to test the batch file. If I find problems, an EXIT

command both restores the environment and returns me to Word where I can make the necessary changes.

Environment considerations when shelling out from Windows

With Windows, you can shell out to perform your environmental experimentations without having to worry about running out of environmental memory. Open the .PIF file for your DOS session using the .PIF editor that comes with Windows. One of the inputs is a space for "Optional Parameters." In this space, enter `/E:800`—only change the 800 to the number of bytes of environmental space you need. Now, when Windows runs DOS, you will have enough environmental space. If you ever run out, just change the number to a higher value.

Using the environment in a batch file

The environment is commonly used as an common exchange ground—for one batch file to place information into the environment for another batch file to use. Looking at this in more depth requires that we understand both the errorlevel and logic testing, so we'll postpone this topic until later in Chapter 18.

Batch files can use environmental variables just like replaceable parameters, including using them as filenames or even commands. This allows environmental variables to be used as a shortcut.

☑ Remember: Batch files can use environmental variables just like replaceable parameters.

Imagine that you're writing a large database program using Quicksilver. You usually work on one program for most of the day, trying to get out all the bugs. You use batch files to start Microsoft Word in order to edit your program and to run Quicksilver in order to compile the program. If you store the name of the program in the environment as Compile, then the batch files can automate much of the work for you.

A batch file segment to compile the program for you, called COMPILE.BAT, would look like this:

```
@ECHO OFF
DB3C -A %COMPILE%
DB3L %COMPILE%
```

The `DB3C -A` command runs the Quicksilver compiler to produce the .OBJ file. It uses the COMPILE environmental variable so that the user doesn't have to enter the name of the program each time. The `DB3L` command runs the Quicksilver linker to produce the .EXE file and again uses the COMPILE environmental variable.

A similar batch file segment, called WORD2QS.BAT, could be written to similarly automate loading the program into Microsoft Word for editing. It would look like this:

```
@ECHO OFF
WORD %COMPILE%
```

Because Word defaults to ASCII mode when editing an ASCII file, no special switches are required.

Summary

The environment is a portion of memory set aside for storing system-wide variables—much like a scratch pad for the operating system.

There are a number of special metacharacters for creating fancy prompts.

The path is a list of subdirectories for DOS to search for a program to run.

The COMSPEC environmental variable tells DOS where to find COMMAND.COM.

The prompt, path, and COMSPEC are stored in the environment.

The size of the environment is limited but can be expanded using DOS commands.

The values of environmental variables are lost when the computer is turned off or rebooted.

A copy of the environment is provided to each program, but any changes made to it are lost when the program terminates.

The SET command is used to store data in the environment. It also erases or changes existing data in the environment.

Batch files—but not the command line—can access the contents of the environment by surrounding the environmental variable name with percent signs.

Batch files can display environmental variables, combine them together, transfer values between them, and even use them as commands.

Programs give you very little free environmental space when you shell out to run other programs.

You can increase the environmental space available when you shell out of a program by creating a dummy environmental variable before you start the program and then deleting the dummy environmental variable when you shell out.

You can increase the available environmental space when shelling out of Windows by editing the DOS .PIF file.

When you return to a program after shelling out to DOS, the environment is restored to its condition prior to your shelling out.

Batch files can use the environment to pass information between themselves.

Batch files can use the environment to store data that will be accessed by more than one batch file or that a single batch file needs to access each time it runs.

Errorlevel

While a nice place for batch files, the environment is a very inhospitable place for programs to store information to communicate with each other. Each program is provided with a copy of the environment when it's first loaded. However, DOS removes as much of the free space from that copy as it can.

When a program reads from or writes to the environment, it's dealing with its copy of the environment and not with the original. When the program terminates, its copy of the environment—along with all the changes the program made to it—are destroyed.

The errorlevel value was provided to overcome this problem. The errorlevel is a single byte of memory so it can only contain any single character. Normally, the content of errorlevel is referred to using its ASCII value rather than its character representation. ASCII values 0–255 are therefore possible. That is why the errorlevel charts below contain numbers rather than characters.

When a program terminates, it can store a number in the errorlevel to indicate its status at termination. When programs use errorlevel, it's the usual rule to set it to 0 for a successful execution and to higher values to indicate different types of errors.

The errorlevel isn't stored in the environment, so changes made to it aren't lost when a program terminates. Unlike the environment, there's only one copy of errorlevel, and every program changes the same one. The errorlevel doesn't change unless a program changes it; you start another program or you reboot, so you can perform multiple tests on it. However, the errorlevel is automatically reset to 0 by DOS each time you start a new program, so each time a program runs, the prior errorlevel value is lost.

You don't need the errorlevel when running programs from the command line since you will see any error messages. However, a batch file would not "see" an error message so errorlevel gives them a way to respond to errors.

The DOS commands that support errorlevel are: BACKUP, CHKDSK, CHOICE, DISKCOMP, DISKCOPY, KEYB, REPLACE, RESTORE, SETVER and XCOPY. TABLES 14-1 through 14-8 list the errorlevel values returned by BACKUP, DISKCOMP, DISKCOPY, KEYB, REPLACE, RESTORE, SETVER and XCOPY, along with their associated meaning. CHKDSK returns a zero if no errors are found and a 255 otherwise. The errorlevel returned by CHOICE depends on how you configure it and this topic is discussed in detail in Chapter 19.

Like DOS commands, only a few commercial programs use errorlevel. If a commercial program uses errorlevel, it will be listed in the program documentation.

While DOS makes little use of errorlevel, it is critical that you become familiar with errorlevel. Almost every batch utility uses errorlevel to communicate results back to the batch file. You will see this in action with the Batcmd utility included with this book. Batcmd is examined in detail in Chapter 19 and Chapter 25.

☑ Remember: Almost every batch utility uses errorlevel to communicate results back to the batch file.

Summary

Errorlevel is a single byte of information that is common to all programs and batch files. It is used by the vast majority of batch file utilities to report their results.

Table 14-1
Errorlevel values for the BACKUP command.

Value	Meaning
0	Backup ran normally.
1	Backup did not find any files matching the command line parameters to back up.
2	A file-sharing problem caused some files not to be backed up.
3	The user terminated the backup with a Ctrl-Break or Ctrl-C.
4	Backup encountered some other form of error not listed above.

Table 14-2
DISKCOMP errorlevel values.

Value	Meaning
0	The disks are the same.
1	The disks are different.
2	The user pressed Ctrl-C.
3	A hard error occurred.
4	An initialization error occurred.

Table 14-3
DISKCOPY errorlevel values.

Value	Meaning
0	The copying was successful.
1	A reading or writing error occurred.
2	The user pressed Ctrl-C.
3	A hard error occurred.
4	An initialization error occurred.

Table 14-4
KEYB errorlevel values.

Value	Meaning
0	Keyboard definition file successfully loaded.
1	Invalid code, code page, or syntax.
2	Keyboard definition file missing or defective.
3	Not used.
4	An error occurred accessing console.
5	The specified code page has not been prepared.

Table 14-5
REPLACE errorlevel values.

Value	Meaning
0	REPLACE was successful.
1	Not used.
2	Source file missing.
3	Source or destination path missing.
4	Not used.
5	You do not have access to the requested files.
6	Not used.
7	Not used.
8	Not enough memory.
9	Not used.
10	Not used.
11	Invalid syntax.

Table 14-6
Errorlevel values for the RESTORE command.

Value	Meaning
0	Restore ran normally.
1	Restore did not find any files matching the command line parameters to restore.
2	A file sharing problem caused some files not to be restored.
3	The user terminated the restore with a Ctrl-Break or Ctrl-C.
4	Restore encountered some other form of error not listed above.

Table 14-7
SETVER errorlevel values.

Value	Meaning
0	SETVER successful.
1	Invalid switch used.
2	Invalid file name used.
3	Not enough memory.
4	Version number format invalid.
5	Specified version table entry missing.
6	SETVER.EXE file missing.
7	Invalid drive.
8	Too many command line parameters.
9	Command line parameters missing.
10	Error reading SETVER.EXE.
11	SETVER.EXE corrupted.
12	This version of SETVER.EXE does not support a version table.
13	Not enough space for another entry.
14	Error while writing to SETVER.EXE.

Table 14-8
Errorlevel values for the XCOPY command.

Value	Meaning
0	Xcopy ran normally.
1	Xcopy did not find any files matching the command line parameters to copy.
2	The user terminated the copy with a Ctrl-Break or Ctrl-C.
3	Not used.
4	Xcopy ran out of memory, Xcopy ran out of disk space, the user specified an invalid drive or the user used invalid syntax.
5	Xcopy encountered an error while writing to the disk.

15 DOS punctuation

DOS comes with some pretty cryptic commands. That is the major reason shells and "graphical user interfaces" are so popular. However, its punctuation is far more cryptic than the commands. If you have ever tried to direct input from one file into a filter and then direct the output to another file, then you know what cryptic means. Because punctuation is often used in batch files, I've devoted this chapter to helping you understand the topic.

Piping & redirection

Normally, you enter input from the keyboard and output goes to the screen, printer, or disk drive. This input and output is often called I/O. DOS has the ability, however, to redirect I/O to and from nonstandard locations. This is called redirection or piping. When you cause input to come from a nonstandard location, it's "input redirection" and when you cause output to go to a nonstandard location, it's "output redirection." Anything else, which usually involves taking output from one program and using it as input for another program is called "piping."

Remember: Input and output redirection and piping cause DOS to handle file inputs and outputs differently. ☑

All of the redirection and piping features of DOS are very similar to analogous UNIX features. They aren't well understood, however, because there were no similar features in CP/M, an operating system for earlier computers and from which many of the features of DOS are derived.

DOS maintains two file handles for I/O, with file handle 0 reserved for input and 1 reserved for output. When you boot, they are both automatically mapped to the console. If you load ANSI.SYS or any Ansi-clone in your CONFIG.SYS, then the

output file handle is remapped to Ansi. What redirection and piping really does is replace one of these standard file handles with a different file handle for I/O processing.

The available piping symbols

There are four symbols used for either redirection or piping:

> The greater-than symbol causes output from one location to flow to another location. Many times, it's treated as input by the receiving program. When the alternative location is a file, this causes the file to be overwritten if it exists. For this reason, you must be very careful when using a greater-than symbol in a batch file. You should always test first, using an IF-test, before piping to a file. If the file exists, you can issue an error message and stop, rename the existing file, or pipe to an alternative file—after checking to make sure that file doesn't exist.

>> The dual greater-than symbol is used to route output to alternative locations. It's the same as the greater-than symbol, except that, when the alternative location is a file, dual greater-than piping causes the messages to be appended to the end of the existing file. This is a safer alternative if you are trying to store error messages or other information in a file. Because the dual greater-than symbol is really just a special case of the greater-than symbol, we will just consider the greater-than symbol for most of our discussion.

< The less-than symbol causes a program to get its input from the specified device or file.

¦ Instead of creating a permanent file like greater-than or dual greater-than symbol, the pipe symbol creates a temporary file that's read and erased by the following program. A common use is in the command `DIR¦MORE`. Another use of pipe symbol is to pipe data that a program needs to run into that program. For example, `ECHO Y¦DEL *.*` will erase all the files in the current directory without asking. That's because the DEL command requires you to confirm the deletion when deleting all the files, but that confirmation is piped into the DEL command using the `ECHO Y` command. Basically, the pipe symbol means that you're using the output of what's on the left as input to what's on the right.

Devices to pipe to/from

Generally, redirection and piping involves either a DOS system device driver or a file name. DOS system device drivers are device drivers that are automatically created by the system. They are as follows:

AUX	This is an auxiliary device that is normally treated as COM1.
CLOCK$	This is the real-time clock.
COM#	This is the serial port. COM1 is the first port, and COM2 is the second. You can also use AUX as the default serial port.
CON	Depending on the usage, this is either the screen or the keyboard. When used as the target, it's the screen. When used as the source, CON is the keyboard.
LPT#	This is the printer port. LPT1 is the first port, and LPT2 is the second port. You can also use PRN to specify the default printer port.
NUL	This is an output device that causes any output sent to it to be discarded.
PRN	This is the printer and is normally mapped to LPT1.

If the target of a redirection or pipe is not one of these, then DOS knows it is a file name.

How DOS handles a pipe

When DOS encounters a less-than symbol, it means a program is going to receive its input from the alternative device specified by the pipe. DOS then closes the standard input device and opens the alternative one specified on the command line because only one input device can be open at a time. That is why, when inputs are redirected into a program from a file, that file must contain all the inputs required by the program. Because the file is the only active source of inputs, no others are available.

The receiving program (the one on the left of the less-than symbol) does not know data is being redirected to it. All it knows is data is coming in from the input device. DOS handled altering the input device before turning control over to the program.

The most common use of input redirection is to construct small programs from magazine articles. The article will have a small script you create with an editor; then, once you are very sure the script is correct, you redirect it into DEBUG with a command like `DEBUG<SCRIPT`. Because everything has to come from the script file, you now know why the magazines are particular to warn you about typing every line, even the blank ones that will represent just a Return press to DEBUG.

You can also use this technique to bypass questions asked by DOS utilities by redirecting answers to them. For example, to bypass the question about formatting another disk in FORMAT, create a text file containing the character "n" and a return, and then format a disk with the command `FORMAT A:<N.TXT`.

Output redirection works in a similar fashion—except, of course, that it's the output file handle that gets replaced rather than the input file handle. For example, if you issue the command `COPY *.* A:` to copy some files to the A drive, you will expect to see each filename scroll by on the screen as it was copied and then to see the DOS "n file(s) copied" message when copying was done.

Now, change the command to

```
COPY *.* A: > NUL
```

and you'll see absolutely nothing extra appear on the screen as the command works. And when it's done, all you will see to indicate it is the return of your prompt! Don't worry, though: The files were still copied. The ">NUL" told DOS to turn off the *standard* output device—the console or ANSI—and turn on the alternative output device—namely NUL in this case. None of this affected the actions of the COPY command itself. This technique is a good way of hiding these sorts of DOS messages from new users in batch files.

If you had wanted to log these messages, you could have redirected them to a file with ">FileName" if it were a one-time log or with ">>FileName" if you did not want to overwrite the log file first. The second approach is particularly useful for accumulating multiple messages in a single file.

When DOS encounters a pipe symbol (¦), it takes the output of one command and passes (or pipes) it on to another command. For example, we can use piping to display directory information one screen full at a time. (Yes, I know DIR/P will do this, but it's a nice easy example.) First, we could get a file listing all our files with

the command DIR>FILE. This would direct the output of the DIR command into a file named FILE. Next, we could send this file through the MORE filter (covered a little later) with the command TYPE FILE > MORE and we would get exactly what we want.

However, that takes two steps and DOS makes it easier. The command DIR¦MORE does the same thing. DOS runs the DIR command, stores its output in a temporary file, and then passes that temporary file over to MORE for display. DOS uses one or more temporary files in the process and these get created prior to the DIR command, so you might see a strange 0-length file or two if you use this command. For example, I just ran it on my computer and saw "AFALBJAE" and "AFALBJAK" in the listing. Don't worry: DOS erases them automatically for you.

Piping and redirection must use DOS

As you experiment with redirection and piping, you are going to find circumstances where it looks like you've done everything right, but the command just does not work and all you end up with is a 0-length file. For example, start Lotus with the command 123>LOG; no matter what you do in Lotus, when you exit, LOG will be a 0-length file! Start Lotus with the command 123<LOG and Lotus will still expect input from you and will not accept it from the LOG file—which must exist, by the way, or DOS will object and not run the command.

There is nothing wrong here with either DOS or Lotus, and there is no problem with the way the commands are constructed. Rather, there is confusion about what redirection and piping does.

For output, DOS offers several character output services or programs that can use video BIOS routines, or they can write information straight to the video buffer. As you might expect, these latter methods are faster, and most programs use them. For input, programs can let DOS read the keyboard for them or they can take the faster approach of using interrupt 16h BIOS services. Output can only be piped when it goes through the standard DOS output device, and input can be piped only when it comes from the DOS standard input device. Because most programs bypass these, most programs are immune to piping. However, most DOS utilities and command-line utilities do use standard DOS I/O, so you can pipe these. When a properly constructed pipe fails to work, the flaw is usually that the program being piped bypasses standard DOS I/O.

This also explains why you cannot use redirection or piping to capture data intended for the printer into a file. Because the printer is not the standard DOS output device but an alternative one, piping does not affect it. To capture data intended for the printer, you need a memory resident program.

Filters to use in pipes

DOS includes three "filters" you can use to manipulate the data coming out of or going into a pipe:

FIND	Searches each line of a file for specified text.
MORE	This displays one screen of information, pauses until you press any key, and then displays another screen of information. This continues until all the information has been displayed. MORE can't scroll backwards. The more common usages are DIR¦MORE or TYPE FILE¦MORE.
SORT	This sorts the file (or input) into ASCII order.

Problems with redirection and piping

Redirection and piping can cause problems, with the major one being a "hung" computer. If you use piping to tell a program to get its input from a file, then that file must contain all the input the program requires—as mentioned above. If it doesn't, you'll either have to reboot or else press Ctrl-Break; either of these acts aborts the program you were running. In addition, redirection and piping can get around the usual warning. For example, the command

```
ECHO Y¦DEL *.*
```

will delete all your files without first asking you if you want to because the ECHO command supplies the necessary response. Those warnings are generally in place for a good reason, and it isn't a good idea to avoid them. Also, redirecting DOS error messages doesn't always work. Another problem crops up when you specify an invalid command, such as:

```
DATE < NUL
PROGRAM < LPT1
```

In the first example, NUL is used as an input device, but NUL can never supply input. In the second case, the printer is used for input and, like NUL, can't supply input. DOS, unfortunately, won't spot either of these problems; the programs or commands will wait forever for a response (or until you press Ctrl-Break or reboot).

Finally, you can destroy data by using redirection and piping improperly. For example, if you wanted to take the data in FILE.DAT and sort it using SORT.EXE, you might issue the command:

```
SORT < FILE.DAT > FILE.DAT
```

This tells DOS to run SORT.EXE and supply it with input from the file FILE.DAT. When SORT.EXE finishes, it's supposed to write the results back to the FILE.DAT file. However, what happens is that FILE.DAT gets erased!

When DOS goes to process a command line or batch file command, the first thing it does is replace any environmental variables or replaceable parameters with their values. There are neither in the above line. The next thing it does is check for redirection and piping and, if it finds any, it initializes any files. Because the above command will redirect data into FILE.DAT, DOS "creates" FILE.DAT and in the process erases the existing copy. Because all this happens prior to DOS actually running the command line, data has not yet been read out of FILE.DAT and redirected into SORT.

This very "feature" is a handy way to create 0-length files if you need them. A 0-length file is a file with a directory entry and valid name but no contents (which means it doesn't take up any disk space). By typing a nonexistent file with a command like `TYPE NOFILE>TARGET`, DOS goes ahead and creates the TARGET file; but because DOS has nothing to put into that file, it remains a 0-length file.

The plus sign is used to perform file concatenation, which is a fancy name for combining two or more files together. For example, the command

```
COPY File1+File2+File3 File4
```

will copy the first three files into one larger file, *File4*. This can cause problems if either of these first three files contains an end-of-file [EOF] marker; most editors won't allow you to go beyond the point in a file where the first EOF marker is located.

You can also use file concatenation to append information to the bottom of a batch file without loading an editor. For example, the command

```
COPY CONFIG.SYS+CON
```

copies the current contents of the CONFIG.SYS file plus whatever you type after issuing this command (press F6 or ^Z to quit entering text) into a new version of the CONFIG.SYS file.

You should use this approach to appending text with care. If the file you are appending text to is large, the operation might not work properly. Plus, if the file doesn't end with a return, you must first press Return, or else the text you enter will be appended to the last line of the file.

: The colon is used to name lines in a batch file. This is explained in detail in Chapter 12.

- DOS 3 and later will usually display a blank line in response to an ECHO command, so you might see this in batch files you receive from others. This trick also works under DOS 6, so you don't need to modify these batch files to use them.

@ The ampersand is used to turn off command echoing for a single command in a batch file regardless of the global status of command echoing. This is covered in Chapter 5.

% The single percent sign is used to mark replaceable parameters in a batch file, as covered in Chapter 10. It is also used to name the variable in a For-loop when issued from the command line rather than in a batch file. This usage is covered in Chapter 12.

%% Dual percent signs are used to name the variable in a FOR-loop when the FOR-loop is used as part of a batch file. See Chapter 12.

%Variable% Percent signs surrounding a variable allow the value of that variable to be used in a batch file. This usage is covered in Chapter 13. This command does not work from the command line.

= The single equal sign is used to assign values to environmental variables and to erase those values. See Chapter 13.

== Dual equal signs are used in IF-tests to test to see if two strings are equal. See Chapter 16.

Summary

By using piping, DOS can get input from alternative sources as well as sending output to alternative locations.

The greater-than sign causes DOS to pipe output to an alternative location. When that location is a file, the file is overwritten.

Dual greater-than signs also cause DOS to pipe output to an alternative location. When that location is a file, the file is not overwritten.

The less-than sign causes a program to obtain its input from an alternative location.

The pipe symbol causes the output of one program to be sent to another program without creating a permanent file.

Devices that can be used as alternative sources for pipes include the console (CON), the printer (LPT or PRN), the serial port (COM), nothing (NUL), and a file.

DOS includes three filters that can work with pipes: Find for locating specific text, More for displaying text one screen at a time, and Sort for placing data in order.

Improper use of piping can cause files to be overwritten.

The plus sign is used to perform file concatenation.

The colon is used to name lines in a batch file.

A period directly following an ECHO causes MS-DOS to display a blank line in a batch file.

The ampersand is used to turn off command echoing for a single command in a batch file regardless of command echoing being turned on.

The single percent sign is used to mark replaceable parameters in batch files and for the variable name in a For-loop issued from the command line.

Dual percent signs are used for the variable name in a FOR-loop issued in a batch file.

Surrounding an environmental variable name with percent signs allows the contents of that variable to be used in a batch file.

The equal sign is used to assign values to environmental variables and to erase those values.

Dual equal signs are used in If-tests to test to see if two strings are equal.

16 Logic testing

Basically, logic testing in a batch file means that you should be able to specify a statement and then have the batch file react differently depending on whether the statement is true or false. The IF-test allows a batch file to accomplish this. The basic syntax of the IF-test is

```
IF Statement Action
```

If the statement being tested is false, then the batch file will skip to the next line and continue processing. If the statement is true, then the batch file will process the rest of this line.

The *Action* portion of the IF-test can be a GOTO command. Using GOTO allows the batch file to take different paths through the batch file—and thus do different things—depending on the conditions the batch file finds while it is running.

Logic testing can take one of four forms:

- Are two strings of characters the same?
- What is the errorlevel?
- Does a file exist?
- Does a subdirectory exist?

When you are comparing two strings of characters, you can acquire these strings from several different sources. They could be

- hardwired into the batch file
- replaceable parameters
- from the environment

Are two values the same?

Batch files can only test to see if two character sets are the same; they cannot perform numeric calculations to see if two equations are equal. A set of characters is called a *string*, and comparing two strings is called a *string comparison*. The format of the IF-test when used for string comparisons is

```
IF String1==String2 Action
```

where *String1* is the source of the first string, *String2* is the source of the second string, and *Action* is the command to be formed when the two strings are the same. Notice that two equal signs are used—not one. If you forget and use a single equal sign, DOS will give you a "syntax error" error message.

One of the strings in this comparison almost always comes from either replaceable parameters or the environment—with replaceable parameters being far the most common source. The other string is generally hardwired into the batch file. I generally place the replaceable parameter on the left side of the two equal signs and the hardwired value on the right side.

One of the things we will use this for throughout the remainder of this book is to add help to our batch files. We will use /? to be consistent with DOS 6, but let's begin by looking at the problem associated with using the older form of /H because it's a generic problem.

When the user starts the batch file with a /H, we want to briefly display a help message and then exit the batch file. You might think that the command to do this is

```
IF %1==/H GOTO HELP
```

and this indeed might work. However, it suffers from two problems—one minor and one more serious.

The minor problem is that string comparisons in a batch file are case-sensitive. Recall that a batch file simply replaces the replaceable parameters with the values entered on the command line. When the user starts this batch file with /H, this test becomes

```
IF /H==/H GOTO HELP
```

which is true. However, when the user starts the batch file with /h, this test becomes:

```
IF /h==/H GOTO HELP
```

which is not true. The solution to this problem is to test for both possible values, so the test becomes

```
IF %1==/H GOTO HELP
IF %1==/h GOTO HELP
```

In this case, if the user entered /H, the first test passes and the batch file jumps to the help section without ever reaching the second test. If the first test fails because the user entered /h, the batch file runs the second test and now finds the two strings equal, jumping to the help section. Of course, if the user enters some other replaceable parameter, both tests would fail and the batch file would continue without executing either GOTO command.

☑ Remember: String comparisons in a batch file are case-sensitive.

While it's not difficult to check the different capitalizations for a single letter, it can become burdensome when multiple letters are involved. For example, if the first replaceable parameter was "daily," "weekly," or "monthly," then it would take nine IF-tests to test for just the likely capitalization combinations of all-lowercase, all-uppercase, or first letter capitalized. The disk that comes with this book includes a utility to deal with this problem, which is explored deeper in Chapter 25.

Also, recall from Chapter 10 that when there is not a space between the batch file name and the slash on the command line, DOS *does not* split the /H apart from the batch name. To make matters worse, DR DOS will split these apart—making it difficult to write batch files that run under both operating systems. We will look at possible solutions to this problem when we examine adding help in Chapter 18.

When you compare strings where one or both of the values come from replaceable parameters, you'll have more problems dealing with the way DOS treats replaceable parameters. Recall from Chapter 10 that everywhere you enter a %1 in the batch file, DOS replaces that %1 with the first replaceable parameter entered on the command line. Thus, if you have the test

```
IF %1==/H GOTO HELP
```

and the user enters a replaceable parameter, then the test works properly—as shown earlier.

Unfortunately, DOS isn't very smart in making these comparisons. When the user does not enter a replaceable parameter, DOS substitutes nothing for %1 and the above comparison becomes

```
IF ==/H GOTO HELP
```

When this happens, DOS thinks you left off part of the IF-test and issues a "Syntax error" error message.

The solution to this minor dilemma is simple: surround the comparison with a set of symbols so DOS will compare something to something. I always use the () symbols. The advantage to surrounding both items to compare with a set of parentheses is that you allow a single test to compare the two strings, while avoiding any problems when one—or even both—of the strings in the comparison are empty. It even allows testing for an empty string because ()==() is a valid test. Thus, the command

```
IF (%1)==() GOTO NONE
```

would successfully test for a blank first replaceable parameter and would return the correct result.

☑ Remember: DOS has problems performing string comparisons when one of the strings is a nul string. The solution is to surround both the strings with parentheses.

In addition to testing the replaceable parameters, we can use the same IF-test to test environmental variables. The only difference is that the environmental variable must be surrounded by percent signs. So the IF-test to see if the prompt is PG is this:

```
IF (%PROMPT%)==($P$G) ECHO Prompt Is $P$G
```

The percent signs around PROMPT tell DOS to replace it with the value of the PROMPT environmental variable so you do not need percent signs around the PG. Of course, like all other IF-tests, this one is case-sensitive.

What is the errorlevel?

As we saw in Chapter 14, errorlevel has room for only one byte, so only an ASCII value of 0 to 255 can be stored in it. Using the statement

```
IF ERRORLEVEL #
```

a batch file can test the contents of errorlevel and make decisions based on that content. Unfortunately, the test isn't straightforward.

The test "IF ERRORLEVEL 5" doesn't test to see if errorlevel is equal to 5, as you might expect from the section on string comparisons. Rather, it tests for an errorlevel of 5 or higher, so any number from 5 to 255 will result in a "true" response to this test.

Remember: The "IF ERRORLEVEL" test is a greater-than-or-equal test and not an equality test. ☑

Because the IF-errorlevel test is a greater-than-or-equal test, you must always test from the highest possible errorlevel value to the lowest. In addition, you must branch away from the testing after the first match. CHECKERR.BAT is a batch file that will test for every possible errorlevel value and display that value on the screen. A small segment of CHECKERR.BAT is shown here:

```
@ECHO OFF
IF ERRORLEVEL 255 ECHO 255
IF ERRORLEVEL 255 GOTO END
IF ERRORLEVEL 254 ECHO 254
IF ERRORLEVEL 254 GOTO END
IF ERRORLEVEL 253 ECHO 253
IF ERRORLEVEL 253 GOTO END
```

—And So On—

Notice how CHECKERR.BAT operates. It first tests for a value of 255. Because the maximum value of errorlevel is 255, this first test is an equality test rather than the normal greater-than-or-equal test. If the value is 255, it displays that information and jumps to the end of the batch file. If the value is not 255, it next tests for 254. Because we know that the errorlevel is not 255 because the batch file has reached this point, the normal greater-than-or-equal test again functions as an equality test. Again, if the value is 254, it displays that value and jumps to the end of the batch file. If not, it tests for 253, 252, and so on down the line. By testing from the highest possible value to the lowest possible value, you can avoid the problems associated with the greater-than-or-equal errorlevel test.

While CHECKERR.BAT is very long, most batch files will be able to test every possible errorlevel value while remaining fairly short. Look back at TABLES 14-1 through 14-3. None of the DOS commands has a possible errorlevel value higher than 5. DOXCOPY.BAT in FIG. 16-1 performs an XCOPY and reports which errors it encountered. Notice that DOXCOPY.BAT tests for all possible errorlevel values from the XCOPY command but is still fairly short.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: DOXCOPY.BAT`
`REM PURPOSE: Run Xcopy Program And`
`REM Report Errors`
`REM VERSION: 1.00`
`REM DATE: May 16, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`XCOPY %1 %2 %3 %4 %5 %6 %7 %8 %9`	Perform the XCOPY command. Any replaceable parameters not used by the user are ignored since they are treated as nul by DOS.
`IF ERRORLEVEL 5 GOTO 5`	Test for the largest possible errorlevel value first and jump to an error-handling section if this is the errorlevel.
`IF ERRORLEVEL 4 GOTO 4`	If the errorlevel was not five, then test for the next highest possible errorlevel value and jump to an error-handling section if this is the errorlevel.
`IF ERRORLEVEL 2 GOTO 2` `IF ERRORLEVEL 1 GOTO 1`	Test for the errorlevel values 1-2 in decreasing order and jump to an error-handling section if either is the errorlevel value. XCOPY does not use an errorlevel value of three.
`GOTO 0`	If the batch file reaches this point, then the errorlevel must be zero so jump to the appropriate section.

`:0`
`  ECHO Normal Completion`
`GOTO END`
	Section for displaying the appropriate message when the errorlevel is zero. After displaying the message, it jumps to the end of the batch file.

`:1`
`  ECHO No Files Found To Copy`
`GOTO END`
	Section for displaying the appropriate message when the errorlevel is one. After displaying the message, it jumps to the end of the batch file.

`:2`
`  ECHO User Pressed Ctrl-Break`
`GOTO END`
	Section for displaying the appropriate message when the errorlevel is two. After displaying the message, it jumps to the end of the batch file.

16-1
DOXCOPY.BAT runs XCOPY and then displays the appropriate error message.

Batch File Line	Explanation
:4 ECHO Not Enough Memory ECHO Or Not Enough Disk Space ECHO Or Invalid Drive Specified ECHO Or Syntax Error GOTO END	Section for displaying the appropriate message when the errorlevel is four. After displaying the message, it jumps to the end of the batch file.
:5 ECHO Disk Write Error GOTO END	Section for displaying the appropriate message when the errorlevel is five. After displaying the message, it jumps to the end of the batch file.
:HELP ECHO Run Xcopy Command ECHO And Display Error Message ECHO Uses DOS 5 Errorlevel Values GOTO END	Section for displaying the appropriate message when the user requests help. After displaying the message, it jumps to the end of the batch file.
:END	Label marking the end of the batch file.

16-1
Continued.

Does a file exist?

The third possible type of IF-test tests to see if a file exists. The format for the command is

```
IF EXIST Filename Action
```

so the command

```
IF EXIST C:\AUTOEXEC.BAT ECHO AUTOEXEC.BAT File Exists
```

would check to see if your AUTOEXEC.BAT file existed in the root directory of the C drive and display a message if it did.

The IF-test will not have any problem testing to see if a file exists, even if the replaceable parameter used in the test does not have a value. Thus, the test

```
IF EXIST %1 Action
```

does not need anything special to protect it in case the user does not enter a replaceable parameter. If fact, putting the replaceable parameter inside parentheses would cause the test to always fail.

The IF-test can use wildcards in the test, so the test

```
IF EXIST C:\BAT\*.BAT ECHO Batch Files Found
```

would be true if at least one .BAT file existed in the C:\BAT subdirectory.

If no path is specified in the IF-Exist test, it only looks for the file in the current subdirectory on the current drive. If you like, you can specify the full path to the file, as is the case in the earlier tests.

Does a subdirectory exist?

The EXIST command is not designed to check to see if a subdirectory exists, but you have two different tricks that can deal with that. First, the command

```
IF EXIST C:\TESTSUB\*.*
```

would check for any file in C:\TESTSUB, and this test would only pass if the subdirectory existed. However, it would not pass if the subdirectory existed but was empty. If you modify this test slightly to

```
IF EXIST C:\TESTSUB\NUL
```

this command uses the DOS device NUL to test the subdirectory where the test passes, even if the subdirectory is empty.

Earlier batch file revisited

Back in Chapter 10, we looked at three batch files: A.BAT, RONNYMD.BAT, and TOA.BAT. A.BAT worked fine, but the other two had problems. We've looked at TOA.BAT a couple of times since then, but it still has some problems. We have enough tools now that we can revisit these three files and put them into final form.

A.BAT

When last we examined A.BAT, it looked like this:

```
@ECHO OFF
REM NAME:    A.BAT
REM PURPOSE: Directory Of A-Drive
REM VERSION: 1.00
REM DATE:    May 12, 1992
DIR A:%1 /P
```

This batch file runs fine, but we can make it better. We'll add command line help to this batch file; and we'll also assume that if the user enters a %2, then the user wants to specify the switches, so we'll leave off the /P. In writing the command line help, we'll assume the batch file will not run under MS DOS; then we can ignore the problem mentioned in Chapter 10 of the /? being part of %0.

The resulting, and final, version of TOA.BAT is shown next:

```
@ECHO OFF
REM NAME:    A.BAT
REM PURPOSE: Directory Of A Drive
REM VERSION: 2.00
REM DATE:    May 12, 1992
IF (%1)==(/?) GOTO HELP
IF (%2)==() DIR A:%1 /P
IF (%2)==() GOTO END
DIR A:%1 %2 %3 %4 %5
:HELP
 ECHO Performs A Directory Of The A Drive
 ECHO Allows You To Specify Files And Switches
 ECHO On The Command Line
GOTO END
:END
```

This new version has a line added at the top to jump to a help section if the user requests help plus four lines to display that help. It has an IF-test added to the front of the old DIR line and only performs that line when the user entered a single replaceable parameter. The next line exits the batch file if the user entered a single replaceable parameter. Finally, it performs a directory using five replaceable parameters. It only reaches this point if the user entered two or more replaceable

parameters; and if the user entered less than five, the extra ones in the batch file are ignored.

When last we examined RONNYMD.BAT, it looked like this:

RONNYMD.BAT

```
@ECHO OFF
REM NAME:     RONNYMD.BAT
REM PURPOSE: Make And Change To A Subdirectory
REM VERSION: 1.00
REM DATE:     May 12, 1992
MD %1
CD %1
```

The major problem with this batch file is it does not confirm that the user entered the name of a subdirectory to create. A minor problem—which we can now completely solve—is it also fails to check to see if the user specified a subdirectory that already exists.

Checking to see that the user entered a replaceable parameter is fairly easy. Checking to see if the subdirectory name the user specified already exists follows the technique shown earlier. The resulting, and final, version of RONNYMD.BAT is shown next:

```
@ECHO OFF
REM NAME:     RONNYMD.BAT
REM PURPOSE: Make And Change To A Subdirectory
REM VERSION: 2.00
REM DATE:     May 12, 1992
IF (%1)==(/?)    ECHO Makes And Changes To A
IF (%1)==(/?)    Subdirectory
IF (%1)==(/?)    GOTO END
IF (%1)==()      ECHO No Subdirectory Specified
IF (%1)==()      GOTO END
IF EXIST %1\NUL ECHO %1 Subdirectory Exists
IF EXIST %1\NUL GOTO END
MD %1
CD %1
:END
```

The most interesting aspect of this modified version is the way it handles help and errors. Rather than jumping to a special section, it uses two IF-tests: the first displays a brief message and the second jumps to the end of the batch file. This approach works when your messages are short. When you need more than one or maybe two lines of messages, you are better off jumping to a special section to handle it.

When last we saw TOA.BAT back in Chapter 12, it looked like this:

TOA.BAT

```
@ECHO OFF
REM NAME:     TOA.BAT
REM PURPOSE: Copy .CMP And .DOC Files To The A Drive
REM           While Deleting .BAK Files
REM VERSION: 3.00
```

```
REM DATE:     May 12, 1992
:TOP
 DEL %1.BAK
 COPY %1.DOC A:
 COPY %1.CMP A:
 SHIFT
GOTO TOP
```

It performs its assigned tasks of deleting the .BAK file and then copying the .DOC and .CMP files to the A drive for the first replaceable parameter. The SHIFT command then brings the next replaceable parameter into the %1 slot and the process is repeated. The one problem is that the batch file does not know when to stop. Even after using all the replaceable parameters, it continues trying to use nonexistent replaceable parameters to copy nonexistent files until the user presses Ctrl-Break.

Fixing TOA.BAT is as easy as adding two lines. First, it needs a label at the end of the batch file to jump to, and it needs an IF-test to jump to the end-label when the batch file runs out of replaceable parameters. The corrected, and final, version of TOA.BAT is shown next:

```
@ECHO OFF
REM NAME:     TOA.BAT
REM PURPOSE:  Copy .CMP And .DOC Files To The A Drive
REM           While Deleting .BAK Files
REM VERSION:  4.00
REM DATE:     May 12, 1992
:TOP
 IF (%1)==() GOTO END
 DEL %1.BAK
 COPY %1.DOC A:
 COPY %1.CMP A:
 SHIFT
GOTO TOP
:END
```

The only thing missing is command line help with the /? switch, and the disk version even has that. Notice that the batch file tests for a blank replaceable parameter at the top of the loop before any processing. That's important, just in case the user starts the batch file without any replaceable parameters. If the logic testing were at the end and the user started the program without a replaceable parameter, the first journey through the loop would generate error messages.

Preview of Chapter 18

Logic testing is so important to successful batch files that, after the review in Chapter 17, Chapter 18 is also devoted to logic testing. The techniques in Chapter 18 are more advanced than this chapter, but often these techniques make a complex problem easier to solve, so they are important techniques to understand.

Summary

The IF-test is used to perform logic testing in a batch file.

The IF-test can test to see if two strings are the same, what the errorlevel is, or if a file exists.

Batch file string comparisons are case-sensitive.

In a string comparison, the variables on both sides of the dual equal signs should be surrounded by parentheses to avoid problems when one of the strings is empty.

An IF-test can test the errorlevel value; however, the test is a greater-than-or-equal test and not an equality test.

Because the IF-test on errorlevel is a greater-than-or-equal test, errorlevel tests must cascade from largest to smallest.

The IF-test can test to see if a single file, a group of files specified with a wildcard, or a subdirectory exists.

Command line help can be added by either jumping to a special section to display the help or by having an IF-test display the help and a second If-test exit the batch file.

17 Applications and problems

It's time to stop and catch our breath. While we have only covered seven chapters since my last questions, we have covered a tremendous amount of material. You now know how to squirt information into a batch file from the command line, how to use that information in a batch file (no matter how much of it was entered on the command line), how to loop, and how to perform logic testing.

It's important for us to take a few moments and make sure that you're up-to-speed before we continue. What follows are three quick problems. Take a few moments and try to work them out. If you get stuck, glance back through the last few chapters. You need to have the skills necessary to solve these problems before you continue through the book; otherwise, you'll run into difficulty.

Problem 4

A friend of yours has a new computer. He is going away for the weekend but wants to run the computer continually while he is gone. He purchased a diagnostic program called Check-It-Out; the command to run the program is CHKITOUT. This program performs a complete diagnostic on the computer. It creates a file containing a full report and sets the errorlevel to 1 when it encounters a problem but does not display any information on the screen.

Your friend has asked you to write him a batch file that could run CHKITOUT continually. Each time CHKITOUT terminates with an errorlevel of 1, the batch file should display an error message on the screen.

Problem 5

Your boss is having problems with her computer. It has a C and a D drive, but sometimes she has to reboot the computer several times before the D drive is operational. She has found a utility program called BOOT.COM that reboots the computer, and she has asked you to write a batch file she can call from her

AUTOEXEC.BAT file to reboot the computer if the D drive is not operational. That way, each time it boots, the computer will automatically reboot itself until the D drive is operational.

Problem 6

You did such a good job on the last problem that your boss has asked you for another favor. It seems that she spends a lot of time on bulletin board systems and acquires a lot of ASCII files to read. Normally, she reads them with the command

```
TYPE file | MORE
```

but she is finding that method to be difficult when she has a lot of files to read. She has asked you for a batch file that loops through and displays all the files that she specifies on the command line. She has also mentioned she is considering you for a raise.

Answers

You have a number of ways to tackle each of these problems. My solutions are included in Appendix A in the back of the book. You don't need to match my answers; if you developed a working solution to the problem, then your solution is just as good as mine and might even be better.

Part four

Advanced batch file topics

Now that you've seen the intermediate batch file commands, it is time to take a look at the advanced commands. By the time you finish this section, you will have seen all the native batch subcommands that are available for you to use in batch files.

Advanced logic testing In Chapter 13, we covered simple logic testing. This chapter expands logic testing to include all the available logic testing commands DOS offers and some more that are supplied by a utility program. — **Chapter 18**

Advanced getting information from users DOS makes it fairly difficult to get information from the user, giving you just replaceable parameters, a single errorlevel-asker, and Ctrl-Break to do the job. This chapter shows how to use a utility program that comes with this book to greatly expand the ability of a batch file to interact with the user. — **Chapter 19**

Advanced screen design 1 Most batch files use a series of ECHO commands to communicate with the user in a very boring fashion. This chapter covers two ways to improve your screens using ASCII files: typing them to the screen, and converting them to a program with a screen compiler. — **Chapter 20**

Advanced screen design 2 This chapter continues where Chapter 17 leaves off by showing you how to use advanced ANSI commands to construct colorful and highly formatted screens. It also briefly covers the many other uses for ANSI. — **Chapter 21**

Batch file subroutines Subroutines are one of the most powerful approaches available to writing complex batch files. This chapter shows you how to use them. — **Chapter 22**

When batch files don't work Even after reading a book as good as this one, you are going to occasionally have problems with the batch files you write. While nothing can eliminate the need for you to work through the logic of your batch file to resolve this problem, this chapter shows you some ways to expedite the process. — **Chapter 23**

Applications and problems Before moving on to the advanced batch subcommands, you should understand the commands and concepts covered so far. This chapter is a quick test. Pass it, and you're ready to move on. — **Chapter 24**

18 Advanced logic testing

In Chapter 16, we began our look at logic testing in batch files. We saw that batch files could test for four conditions in a batch file:

- Whether or not two strings of characters are identical.
- What the errorlevel is.
- Whether or not a file exists.
- Whether or not a directory exists.

When you compare two strings of characters, you can acquire them from several different sources:

- Hardwired into the batch file.
- From replaceable parameters.
- From the environment.

In this chapter, we will examine logic testing in even more detail. Included in this text will be your first look at some of the utilities included on the disk that comes with this book. As you will see, these utilities will make your batch files extremely powerful.

NOT

The NOT modifier is used to reverse an IF-test. If an IF-Exist test is true, then the corresponding IF-Not-Exist test is false. If an IF-Exist test is false, then the IF-Not-Exist test is true. For example, the batch file segment

```
IF EXIST C:\AUTOEXEC.BAT ECHO AUTOEXEC.BAT Exists
IF NOT EXIST C:\AUTOEXEC.BAT ECHO AUTOEXEC.BAT Missing!
```

will tell the user about her AUTOEXEC.BAT no matter if it exists or not.

Inter-batch communications

You can have batch files communicate with each other by placing values into the environment. ACCOUNT1.BAT performs monthly closing. The program CLOSE.EXE sets the errorlevel to show whether or not it ran without errors. ACCOUNT1.BAT needs to communicate that information to ACCOUNT2.BAT. It could depend on the errorlevel, but that could be reset if you ran another program in between the two batch files. To avoid that problem, ACCOUNT1.BAT places a value into the environment. The portion of ACCOUNT1.BAT that handles running the closing program and creating the environmental variable is shown here:

```
CLOSE
IF ERRORLEVEL 1 SET ACCOUNT=NO
IF ERRORLEVEL 1 GOTO END
IF ERRORLEVEL 0 SET ACCOUNT=YES
```

ACCOUNT2.BAT then checks for that value and won't run without a correct setting. The portion of ACCOUNT2.BAT that handles this is shown next:

```
IF (%ACCOUNT%)==() GOTO ERROR
IF (%ACCOUNT%)==(NO) GOTO NO
CLOSE2
GOTO END
:ERROR
 ECHO Run Account1 First
 PAUSE
 GOTO END
:NO
 ECHO Warning: Close Did Not Run Successfully
 ECHO Fix These Errors Before Running Account2
 PAUSE
 GOTO END
:END
SET ACCOUNT=
```

Capitalization is not a problem here because ACCOUNT1.BAT always creates the variables in uppercase. Also note that ACCOUNT2.BAT resets the ACCOUNT environmental variable after running, which keeps ACCOUNT2.BAT from being run twice. This is something that you could not prevent with just errorlevels.

A logical question is "Why didn't CLOSE.EXE set the information into the environment itself?" Well, each program is passed a copy of the environment when it starts. When the program terminates, that copy of the environment is erased. So CLOSE.EXE couldn't set the environment directly because those changes would have been lost when CLOSE.EXE terminated. Because errorlevel isn't stored in the environment, it isn't lost when CLOSE.EXE terminates.

Dealing with capitalization

Managing capitalization in batch files is a real pain. As we said in Chapter 16, just testing for the likely capitalization of three possible replaceable parameters—"Daily," "Monthly," and "Annual"—takes nine IF-tests. That means the code just to handle the most likely capitalizations takes up nine lines, where three would work if capitalization were not a problem.

Even this elaborate scheme will not respond properly to replaceable parameters such as DAIly. There are a number of ways of dealing with the capitalization problem quickly without a lot of extra code. Each method has unique advantages, so we'll look at each in more detail.

GOTO %1

The first way to avoid the capitalization problem is to use the replaceable parameter as a label for the GOTO command because labels are case-insensitive. Keep in mind that DOS always replaces the replaceable parameters with their value. Take a look at this batch file segment:

```
GOTO %1
:DAILY
 DAILY
GOTO END
:MONTHLY
 MONTHLY
GOTO END
:ANNUAL
 ANNUAL
GOTO END
:END
```

If you run this batch file with the command line

```
BATCH daily
```

DOS replaces the `GOTO %1` line with `GOTO daily` and the batch file runs properly—no matter what capitalization you use for daily.

This method is only a partial solution. If the user starts the batch file with an invalid parameter—"yearly," for example—the batch file will abort on the GOTO yearly line with a "Label not found" error message. Because the batch file "bombs" after a GOTO %1 command if the specified label does not exist, no commands are needed under the GOTO %1 other than the sections that are being jumped.

While there is no absolute protection against invalid labels when using this approach, there is partial protection. Each section can use multiple labels with all the labels the user is likely to use. For example, the Annual section might look like this:

```
:ANNUAL
:ANNUALLY
:YEAR
:YEARLY
:ENDOFYEAR
 ANNUAL
 GOTO END
```

This helps prevent the "Label not found" by putting as many possible labels in the batch file as the user is likely to use. Because label lines are not executed, having the batch file jump to the :ANNUAL or the :ENDOFYEAR line has the same effect on the execution of the batch file.

Environmental variables

A second method is to store the selected option in the environment as an environmental variable. The next batch file segment illustrates this:

```
SET DAILY=
SET MONTHLY=
SET ANNUAL=
SET %1=YES
IF %DAILY%==YES GOTO DAILY
IF %MONTHLY%==YES GOTO MONTHLY
IF %ANNUAL%==YES GOTO ANNUAL
GOTO ERROR
:DAILY
 DAILY
GOTO END
:MONTHLY
 MONTHLY
GOTO END
:ANNUAL
 ANNUAL
GOTO END
:ERROR
 ECHO You Entered An Invalid Parameter
GOTO END
:END
```

First, each possible environmental is deleted; then the line `SET %1=YES` creates a single environmental variable with the name of the replaceable parameter and a value of YES.

This method works well for most replaceable parameters and avoids the missing label problem associated with the `GOTO %1` method but has a minor problem of its own. If the replaceable parameter the user enters can be a number, the batch file will end up making unexpected tests. Suppose you wanted to run Lotus 1-2-3, so the user could be expected to use 123 as a replaceable parameter. The `SET %1=YES` line would work properly. In order to run Lotus, you would need a test line of `IF %123%==YES`. However, DOS would translate the %1 portion of this test as the first replaceable parameter.

The path

The path is the only environmental variable DOS converts to uppercase. You can use this to convert a replaceable parameter to uppercase by performing the following steps:

1. Store the current path under another variable name.
2. Set the path equal to the replaceable parameter. Note that you must use PATH= statement and not SET PATH= because DOS will not convert the path to uppercase using the SET PATH= method.
3. Store the replaceable parameter now stored under the path to another variable name.
4. Restore the proper path using the holding variable created in Step #1.
5. Clear out the holding variable created in step #1.

The following file segment illustrates this:

```
SET OLDPATH=%PATH%
PATH=%1
SET VARIABLE=%PATH%
PATH=%OLDPATH%
SET OLDPATH=
IF (%VARIABLE%)==(DAILY)   GOTO DAILY
IF (%VARIABLE%)==(MONTHLY) GOTO MONTHLY
IF (%VARIABLE%)==(ANNUAL)  GOTO ANNUAL
GOTO ERROR
```

This method handles all possible inputs without problem but requires the most environmental space; you must store the path under a different name before you store the replaceable parameter to the path variable, so at this point DOS must store two versions of the path in environmental memory at once. Because most paths are long, this takes a lot of memory.

Run a program

I became so frustrated trying to deal with capitalization that I wrote a program called CAPITAL.EXE to cure the problem once and for all. CAPITAL takes a single word as an input, converts it to all uppercase, and stores the results in the environment under the environmental variable name RONNY. Before we use this program, a couple of notes are in order.

Normally, a program like CAPITAL does not have access to the master copy of the environment. So, if CAPITAL were to just place the results in its copy of the environment, that information would be lost as soon as CAPITAL terminated. That is clearly not what we want. The only remaining "usual" way to pass the information back to the batch file is via the errorlevel, but that value is limited to one byte. CAPITAL avoids the problem by searching through all the computer memory to find the master copy of the environment. It then makes its changes there. This is the same approach taken by most batch utilities that work with the environment.

Second, because CAPITAL expects to store the results in the environment, it will not be able to run if there is not enough free environmental space. Third, CAPITAL only works on one word at a time. Send it more than one word, and it aborts and displays an error message. The next batch file segment illustrates using CAPITAL:

```
CAPITAL %1
IF (%RONNY%)==(DAILY)   GOTO DAILY
IF (%RONNY%)==(MONTHLY) GOTO MONTHLY
IF (%RONNY%)==(ANNUAL)  GOTO ANNUAL
GOTO ERROR
```

Of course, somewhere near the end of the batch file you would want a `SET RONNY=` command to reset this variable once you no longer needed it.

Capitalization conclusion

Each of these approaches has good points and bad points. The one you select really depends on your needs. However, I find that most of my batch files can be written using just the `GOTO %1` and CAPITAL.EXE approach. I use the `GOTO %1`

approach when I have a lot of possible labels and the CAPITAL approach when there are fewer labels.

Multiple IF-tests

Several IF-tests can be combined to handle a complex decision or to help overcome the case problem associated with string comparisons. The format of a two-deep multiple IF-test is

```
IF Statement IF Statement2 Action
```

When the first IF-test fails, control flows to the next line in the batch file as it normally does. When the first IF-statement is true, control passes to the remainder of the command as it normally does—only in this case, the remainder of the command is another IF-test. Now, if the second IF-test fails, control flows to the next line in the batch file; but if the second IF-test is true, the *Action* command is executed. So each IF-test acts as a gate, and the *Action* command is only executed when every IF-test before it is true. If just one of the IF-tests is false, the entire test fails and the *Action* command is not executed.

DOS does not place a direct limit on the number of IF-tests you can combine in this fashion. You can use three or four or even five if you like. However, the command must be shorter than the 127-character limit DOS places on all batch file commands. In addition, the logic must be clear enough that the batch file author and anyone who later needs to maintain the batch file can understand it.

A shorter way to show the errorlevel

Consider the following example:

```
IF ERRORLEVEL 3 IF NOT ERRORLEVEL 4 ECHO Errorlevel Is 3
```

Three conditions are possible: the errorlevel is less than 3, the errorlevel is greater than 3, or the errorlevel equals 3.

When errorlevel is less than 3, the first IF-test fails and the batch file would move on to the next line. When errorlevel is greater than 3, the first IF-test would pass. When an IF-test passes, the batch file executes the remainder of the line—which in this case contains another IF-test. For this second IF-test, the test `IF ERRORLEVEL 4` is true, but the NOT modifier changes the results to false. Thus, the second IF-test fails and the batch file continues on to the next line.

Now, when the errorlevel is equal to 3, the first IF-test passes and again the batch file moves on to the remainder of the IF-test. In this case, the test `IF ERRORLEVEL 4` is false because errorlevel is 3, so this test fails. However, the NOT modifier changes the results to true. Having passed both IF-tests, the batch file moves on the next portion of the line and displays the errorlevel value.

In general, the IF-test

```
IF ERRORLEVEL n IF NOT ERRORLEVEL n+1 Action
```

is only true for an errorlevel value of exactly *n*. Take a look at CHECKER1.BAT, a brief segment of which is shown below. This batch file takes advantage of these multiple IF-tests to considerably reduce the number of lines needed to display all possible errorlevel values over CHECKERR.BAT in Chapter 16.

```
IF ERRORLEVEL 255                                    ECHO 255
IF ERRORLEVEL 254 IF NOT ERRORLEVEL 255 ECHO 254
IF ERRORLEVEL 253 IF NOT ERRORLEVEL 254 ECHO 253
IF ERRORLEVEL 252 IF NOT ERRORLEVEL 253 ECHO 252
```

—And So On—

While CHECKER1.BAT is shorter, it will take much longer to run—especially when the errorlevel is a high number. When CHECKERR.BAT processes a high errorlevel number—say 250—it only has to go through a few IF-tests before it finds a match. For a value of 250, it would go through twelve IF-tests before jumping out of the batch file. However, for each errorlevel value, CHECKER1.BAT processes all 255 of the first IF-tests. In addition, it processes the second IF-test for all errorlevel tests with lower test values than the errorlevel value. That is, for errorlevel test values of 0–250, CHECKER1.BAT must move on to the second IF-test because the first is true. So for 250, CHECKER1.BAT ends up processing 305 IF-tests rather than the twelve CHECKERR.BAT required and so it runs slower.

A very short way to show the errorlevel

Warning: This explanation of a very short way to show the errorlevel is fairly complex and is not critical to your understanding the rest of the material. If you do not fully understand IF-tests and For-loops, you should postpone reading this section until your skills improve.

While writing my *Dr. Batch File's Ultimate Collection* book, I developed an extremely short and fast batch file for displaying the errorlevel. I've found it so useful that I keep it in my utility subdirectory and find myself using it all the time. Because it's so useful, I want to share it with you now.

An easier way to find the errorlevel is to test each of the three digits individually. First, find out if the left digit is a 0, 1, or 2. This only takes one FOR-loop. The next batch file segment does this:

```
FOR %%J IN (0 1 2) DO IF ERRORLEVEL %%J00 SET ERROR=%%J
ECHO Left Digit Was %ERROR%
```

In addition to displaying the left digit, it also stores it in the environment under the environmental variable name ERROR. That is required for the remaining steps.

The next step is to find out if the middle digit is a 0–9. That takes only one FOR-loop as well. The following batch file segment does this:

```
FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL
 %ERROR%%%J0 SET ERROR=%ERROR%%%J
ECHO Left Two Digits Are %ERROR%
```

Notice that the variable ERROR is being built as we go, from left to right. The right portion of the first line looks confusing but DOS interprets it properly. For your information, let's look at this line in more detail.

`FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO` is the typical beginning of a FOR-loop that causes it to loop through the numbers 0–9.

`IF ERRORLEVEL` specifies that, for each FOR-loop, an IF-test on the errorlevel value will be performed.

%ERROR% is replaced by the environmental variable value. That will be the left digit of the errorlevel value—usually a 0.

%%J is replaced by the value of the loop counter variable on each loop.

0 is a placeholder for the last variable, which is necessary because we are testing for the middle variable. 0 is the smallest value, so it is used.

SET ERROR= stores the value of the test in the environment when the test passes.

%ERROR%%%J are environmental variables. When the value is stored to the environment because the IF-test passed, %ERROR is replaced by the current value of the left digit stored in the environment and %%J is replaced by the loop counter value.

Testing for the right digit works the same way. The next batch file segment does this:

```
FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO IF ERRORLEVEL
 %ERROR%%%J SET ERROR=%ERROR%%%J
ECHO ERRORLEVEL is %ERROR%
```

The FOR-loop in this batch file works much like the one just explained.

This method works properly for errorlevel values 0–199, but there is a problem for errorlevel values over 200. There are no restrictions in the batch file, so when the first digit is a 2, the remaining segments end up testing for values 256–299. The maximum errorlevel value is 255, and DOS does not handle tests above 255 properly. As a result, we need some complex branching.

If the first digit is a 2, the batch file must branch to a separate test to make sure the test on the second digit does not exceed 5. If the second digit is a 5, it must branch again to make sure the test on the final digit does not exceed a 5. That way, the batch file never tests for an errorlevel greater than 255. In addition, there is no advantage in using separate batch files for each digit, so all three batch files can be combined into one. The resulting batch file is SAYERROR.BAT in FIG. 18-1.

18-1
SAYERROR.BAT quickly displays the errorlevel and stores its value in the environment for future reference.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: SAYERROR.BAT`
`REM PURPOSE: Show Errorlevel`
`REM VERSION: 1.00`
`REM DATE: November 10, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`SET ERROR=`	Reset the environmental variable to nul.
`FOR %%J IN (0 1 2) DO IF ERRORLEVEL %%J00` `SET ERROR=%%J`	Find out if a) errorlevel>200 b) errorlevel>100 c) errorlevel>0.

Batch File Line	Explanation
`IF %ERROR%==2 GOTO 2`	Jump to a special section if errorlevel is 200 or larger. The maximum errorlevel is 255 and DOS does not handle tests for error-levels over 255 nicely. When testing "IF ERRORLEVEL 260" (for example) DOS subtracts 256 so the number is less than 256, or 4 in this example.
`IF %ERROR%==0 SET ERROR=`	If the hundreds digit is a zero, remove it.

`FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO`
`  IF ERRORLEVEL %ERROR%%%J0 SET`
`  ERROR=%ERROR%%%J`
	As complex as this looks, DOS under-stands it. If the value of ERROR was 1 coming into this test and the FOR loop is on 6, the test line reads "IF ERROR-LEVEL 160 SET ERROR=16" This test sets the value of the tens digit.

`IF %ERROR%===0 SET ERROR=`	If the tens digit is a zero and there is no hundreds digit, remove the zero.

`FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO`
`  IF ERRORLEVEL %ERROR%%%J`
`  SET ERROR=%ERROR%%%J`
	This test sets the value of the ones digit.

`GOTO END`	Jump to the end of the batch file.
`:2`	This marks the beginning of the section that handles errorlevel greater than or equal to 200.
`FOR %%J IN (0 1 2 3 4 5) DO IF ERRORLEVEL` `  %ERROR%%%J0 SET ERROR=%ERROR%%%J`	Test for the tens digit. Since the maximum errorlevel value is 255, the test does not need to exceed 5.
`IF %ERROR%==25 GOTO 25`	If the errorlevel is 250 or larger, jump to a special section because the batch file needs to test only to five.

18-1
Continued.

18-1
Continued.

Batch File Line	Explanation

`FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO`
`  IF ERRORLEVEL %ERROR%%%J SET`
`  ERROR=%ERROR%%%J`
	Test for the ones-digit.

`GOTO END`	Exit the batch file.
`:25`	Label marking the section of the batch file to test error-level values greater than or equal to 250.
`FOR %%J IN (0 1 2 3 4 5) DO IF ERRORLEVEL` `  %ERROR%%%J SET ERROR=%ERROR%%%J`	Test for the ones-digit.
`GOTO END`	Exit the batch file.

`:HELP`
`  ECHO Displays The Errorlevel`
`  ECHO And Stores It To The`
`  ECHO Environment`
`  ECHO All Values Reported`
`  ECHO Correctly`
`GOTO END`
	Section that displays help when the user starts the batch file with a /? or a ? as the first replaceable parameter.

`:END`	Label marking the end of the batch file.

SAYERROR.BAT quickly tests the errorlevel values. For every errorlevel value, the test took under a second. In addition to displaying the errorlevel value, SAYERROR.BAT also stores the value in the environment under the name ERROR. That allows you to retain the errorlevel value while running another program. It also makes testing on the errorlevel value much easier. While the usual errorlevel test of `IF ERRORLEVEL #` is a greater than or equal test, the test `IF %ERROR%==#` is an equality test. This reduces the number of IF-statements required to test for any value in half.

Note: While the operation of SAYERROR.BAT is difficult to follow, its operation is simple. Just enter `SAYERROR` at the command line, and it stores the errorlevel value in the environment under the environmental variable ERROR and displays it on the screen. You can even use SAYERROR.BAT without understanding how it operates.

Setting the errorlevel

Testing the errorlevel value can be difficult. When I first started working with errorlevel IF-tests, I did things like using XCOPY and leaving the drive open to force an errorlevel value into DOS. Then, I received my copy of Batcmd (short for Batch Commands). Batcmd is a program designed to be run from inside a batch file. The general format of the command is

`BATCMD` *`Keyword <Prompt>`*

where *`Keyword`* is a two-letter command abbreviation that tells Batcmd what to do and *`Prompt`* is an optional message to the user to tell him/her what information to enter. We'll talk about Batcmd in more detail later in this chapter, as well as in Chapter 19 and Chapter 25; but for now, I'll just say that one of

Batcmd's nicer features is its ability to force any value into the errorlevel. The command to do that is

```
BATCMD EX #
```

where # is a number 0–255 to be forced into the errorlevel.

Being able to force any value 0–255 into errorlevel makes it easy to test batch files. It is also handy for resetting the errorlevel after an XCOPY or other command has failed and stored a value in the errorlevel. A copy of Batcmd is included on the disk that comes with this book.

Help

Back in Chapter 16, we hinted at batch file help while exploring the capitalization problem associated with IF-test string comparisons. Now is the time to look at batch file help in more detail.

Beginning with version 5, DOS added the ability to get quick summary information about most programs by running the program with a /? switch. When you do this, the program displays summary information and then exits to the command line without doing anything other than displaying help. That way, you can read about a program without having to run it.

We can add this to batch files. The general format in pseudocode is

If the user requested help
Display the help information
Exit the batch file

The second and third line of the pseudocode are easy. You display help information using a series of ECHO commands and you exit the batch file by having a line labeled :END at the end of the batch file and using a `GOTO End` command.

There are basically two ways a user might run a batch file and request help: `BATCH/?` and `BATCH /?`. If you are running DR DOS 5 or 6 exclusively, then you don't have a problem; DR DOS 5 and 6 will automatically split the batch file name from the switch, so %0 would be "BATCH" and %1 would be "/?" no matter which way the user entered the information. However, MS-DOS does not do this. If you run "BATCH/?" under any version of MS-DOS then %0 is "BATCH/?" and %1 is empty.

Warning: DR DOS and MS-DOS handle replaceable parameters differently when the name of a batch file is followed immediately by a slash with information after the slash. DR DOS makes the slash and everything that follows %1, with the next replaceable parameter %2, and so on. Thus, if you enter BATCH/? under DR DOS, BATCH becomes %0 and /? becomes %1. However, MS-DOS treats everything before the first space as %0, so if you enter BATCH/? under MS DOS, BATCH/? becomes %0. This makes it very difficult to write batch files that run under both MS- and DR DOS and accept parameters with a slash.

So, running a batch file called BATCH.BAT under MS-DOS and checking for a /? switch requires the following tests just to test for the most likely capitalizations:

```
IF (%1)==(/?) GOTO HELP
IF %0==BATCH/? GOTO HELP
IF %0==batch/? GOTO HELP
IF %0==Batch/? GOTO HELP
```

And even this will not catch unusual capitalizations. Note that I did not use parentheses around %0; I omitted them because %0 always has a value.

To avoid this problem, I wrote the program NEEDHELP.EXE; it's on the disk that comes with this book. To use NeedHelp, just pass it your %0 and %1 variables with a space between them. It checks to see if the %1 variable or the last two characters of %0 are /?. If so, it sets the errorlevel to 1; otherwise, it sets it to 0. In other words, an errorlevel of 1 indicates the user requested help, while an errorlevel of 0 indicates the user did not request help.

If you do not pass NeedHelp a parameter, it sets errorlevel to 255. It does this because %0 always exists, so not receiving a parameter means you made an error. It does not object to only one parameter because %1 might not have a value.

To use NeedHelp, place these following lines near the top of your batch file:

```
NEEDHELP %0 %1
IF ERRORLEVEL 1 GOTO HELP
```

Now your batch files will work the same under MS-DOS as they do under DR DOS, without you having to worry about how the specific operating system handles command line switches. You also no longer have to worry about the capitalization the user enters the batch name in.

Windows

Occasionally, it would be nice for your batch files to know if they are running under Windows. For example, while the CHKDSK command is safe under Windows, the CHKDSK/f command can destroy your hard disk under Windows. Without knowing if Windows is running, your only choice is to not allow CHKDSK/f from your batch files just in case Windows *is* running.

Windows always creates an environmental variable called windir when it's running, and this variable points to the subdirectory where Windows is located. However, Windows is very tricky about this variable. DOS leaves the contents of your environmental variable alone, but it always converts environmental variable names to all uppercase. However, Windows bypasses DOS and creates the "windir" environmental variable name in lowercase. This keeps you from being able to alter the windir environmental variable yourself—either in a batch file or from the command line. As a result, the normal batch IF-tests to see if the windir environmental variable exists will not work.

To get around this problem, I've included a small program on the disk that comes with this book called ISITWIN.EXE. IsItWin sets errorlevel to 1 if it's running while you are shelled out from Windows, and it sets errorlevel to 0 if it's run from the command line without Windows loaded in memory. That way, you can build Windows-checking into those batch files that need it. IsItWin works by testing on the windir but bypassing DOS for this testing so it can test on a lowercase environmental variable.

The NOT modifier reverses the decision of any IF-test.

Summary

By creating environmental variables, batch files can communicate with each other even when there is a time lapse between each of them running.

One way to deal with capitalization of replaceable parameters is with a `GOTO %1` command.

When using GOTO %1 to deal with capitalization of replaceable parameters, the batch file bombs when the label does not exist. The chance of this can be reduced by using multiple labels.

Another way to deal with capitalization of replaceable parameters is by creating an environmental variable with the name of the replaceable parameter because environmental variable names are always converted to uppercase.

A third way to deal with capitalization of replaceable parameters is to temporarily store the replaceable parameter to the path because the path is converted to uppercase.

A final way to deal with capitalization of replaceable parameters is to run the CAPITAL program included on the disk that comes with this book.

Multiple IF-tests can be combined on a single line.

When multiple IF-tests are combined, the Action command is not executed unless all of the IF-tests are true.

When multiple IF-tests are combined, their length must not exceed 127 characters.

The SAYERROR.BAT batch file on the disk that comes with this book quickly displays the errorlevel as well as storing it to the environment under the name ERROR.

The Batcmd program on the disk that comes with this book allows you to set the errorlevel to any value 0–255.

Batch files that test to see if %1 is /? to display help might not work under MS DOS.

The NeedHelp program on the disk that comes with this book checks to see if the user requested help using the same commands under both MS- and DOS.

When running, Windows creates a windir environmental variable that DOS can display but cannot modify or test on because the name is stored in lowercase.

The IsItWin program that comes on the disk that comes with this book sets errorlevel to 1 if Windows is running and to 0 if Windows is not running.

19 Advanced getting information from users

The DOS errorlevel is one of the most useful items available to batch file authors, yet it is also one of the least used. Part of the reason that it's avoided is because of its idiotic name, which tells the user nothing about what errorlevel actually does and probably scares off a lot of potential users. The other reason errorlevel is rarely used is that prior to DOS 6, DOS did not give you any tools to take advantage of the errorlevel. At best, a very few DOS programs set the errorlevel when they exit.

So far, we have seen only two ways to get information from the user: replaceable parameters, and Ctrl-Break. With replaceable parameters, the user must remember the questions in advance—before running the batch file—in order to enter that information on the command line. That's asking for a lot! I have trouble remembering what batch file I need to run, much less what questions it is going to ask me. Many times, the questions change each time the batch file runs. "Do you really want to delete these files?" takes a lot of different meanings depending on which files the batch file shows me.

Pausing the batch file and asking the user to press any key to continue or Ctrl-Break to stop is one way of getting a yes/no response. It's inelegant but it works. However, because the batch file stops in response to a Ctrl-Break, it cannot perform any other tasks if the user wants to stop this operation.

All in all, the tools DOS 5 and earlier gives you to query the user are a pretty sorry lot! However, the single byte of errorlevel memory gives us a way to dramatically improve the situation and we will see that in a few moments. However, we first need to look at the improvement in DOS 6.

Choices in DOS 6

DOS 6 includes CHOICE.COM, a small "errorlevel-asker" program. "Errorlevel-asker" is a term I coined in my *Batch File Utilities* book to describe programs that ask the user to select a single-character response in a batch file and then make that response available to the batch file via the errorlevel.

The format for using the CHOICE command is:

```
CHOICE [/C[:Keys]] [/N] [/S] [/T[:Default,Time]] [Prompt]
```

/C When used by itself, the CHOICE command causes the user to select between a "Y" and an "n." You can have the user select between any keys you like by using the /C: and then listing the keystrokes to accept. They should be bunched together without anything between them and capitalization does not matter unless you use the /S.

/N Causes CHOICE not to display the valid keys; however, the prompt is still displayed. This might be useful for prompting for a password.

/S Causes CHOICE to be case-sensitive. By default, CHOICE ignores case and treats "a" and "A" equally. That saves you the trouble of having to deal with capitalization. The only time you should need this option is when you need to have the user select more than 36 options (26 letters and 0–9).

/T This specifies a time for CHOICE to wait for the user to press a key and a default key to use if the user does not make a selection in the allowed time. The time is specified in seconds with a value of 0–99.

When you run CHOICE, it shows the prompt you specify followed by a list of allowable characters inside square brackets and separated with commas—unless you use the /N switch, in which case it only shows the prompt. Because the allowable characters are listed after the prompt, you should always end your prompt with a space. Additionally, if you load ANSI.SYS in your CONFIG.SYS file, the prompt can contain the Ansi escape sequences covered in Chapter 20.

When the user makes an invalid selection—one that is not on the list of allowed characters—CHOICE just beeps and continues waiting for an allowable character. Once an allowable character has been selected, CHOICE sets the errorlevel to indicate the position of the character in the allowable list. The first character gets an errorlevel of one, the second two, and so on.

So you can see CHOICE in action, the disk contains a demonstration batch file called PICKDEMO.BAT. Just run it and it will show you some of the power of CHOICE. Just remember that this demonstration requires DOS 6. PICKDEMO.BAT is described in detail in FIG. 19-1.

Batcmd: Batch commands

The idea is a simple one: Use a utility program to ask the user a question and then code the user's response into errorlevel or into the environment. This book includes a program called Batch Commands—Batcmd for short—that makes it easy to get information from the user.

Note: Batcmd makes asking the user questions via the errorlevel a snap.

Batcmd 1.0 was written by Doug Amaral of hyperkinetix—the maker of Builder and Builder Lite—especially for this book. I added a couple of commands and made

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: PICKDEMO.BAT`
`REM PURPOSE: Choice Demonstration`
`REM VERSION: 1.00`
`REM DATE: December 6, 1992`
	Documentation remarks.

`NEEDHELP %0 %1`
`IF ERRORLEVEL 1 ECHO Illustrates Choice Command In DOS 6`
`IF ERRORLEVEL 1 ECHO Requires DOS 6 To Run`
`IF ERRORLEVEL 1 GOTO END`
	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch. If so, displays help and exits.

`ECHO Requires DOS 6 To Run`	Displays an important message.
`CHOICE Do You Have Ansi Loaded?`	Asks the user if Ansi is loaded.
`IF ERRORLEVEL 2 IF NOT ERRORLEVEL 3 ECHO You Said No` `IF ERRORLEVEL 1 IF NOT ERRORLEVEL 2 ECHO You Said Yes`	Tells the user his response.
`IF ERRORLEVEL 1 IF NOT ERRORLEVEL 2 CHOICE ESC[41;30mMake Selection /C:ABC`	If the user has Ansi loaded, ask for a letter using a Choice prompt containing Ansi.
`CHOICE Pick A Letter /C:ABCDEFGHIJKLM NOPQRSTUVWXYZ`	Ask the user to enter a letter. This will accept either case.
`ECHO The Next Choice Will *Only* Accept Uppercase` `ECHO But Try A Lowercase Letter First`	Display a message.
`CHOICE Pick A Letter /C:ABCDEFGHIJKLM NOPQRSTUVWXYZ /S`	Ask the user to enter a letter but use the "/S" switch to force Choice to only use the specified case.
`CHOICE Pick A Number (You Have Ten Seconds) /C:123456789 /T:5,10`	Ask the user for a number and use the "/T" switch to specify a default value and a timeout period.

`IF ERRORLEVEL 1 IF NOT ERRORLEVEL 2 ECHO You Selected 1`
`IF ERRORLEVEL 2 IF NOT ERRORLEVEL 3 ECHO You Selected 2`
`IF ERRORLEVEL 3 IF NOT ERRORLEVEL 4 ECHO You Selected 3`
`IF ERRORLEVEL 4 IF NOT ERRORLEVEL 5 ECHO You Selected 4`
`IF ERRORLEVEL 5 IF NOT ERRORLEVEL 6 ECHO You Selected 5`
`IF ERRORLEVEL 6 IF NOT ERRORLEVEL 7 ECHO You Selected 6`
`IF ERRORLEVEL 7 IF NOT ERRORLEVEL 8 ECHO You Selected 7`
`IF ERRORLEVEL 8 IF NOT ERRORLEVEL 9 ECHO You Selected 8`
`IF ERRORLEVEL 9 IF NOT ERRORLEVEL 10 ECHO You Selected 9`
	Use a series of errorlevel tests to tell the user which number was selected.

`:END`	Label marking the end of the batch file.

19-1
PICKDEMO.BAT illustrates using the DOS 6 CHOICE command.

some minor modifications to the help system to produce the version 1.1 that is included in this book.

Batcmd is a program designed to be run from inside a batch file. The general format of the command is

```
BATCMD Keyword <Prompt>
```

where *Keyword* is a two-letter command abbreviation that tells Batcmd what to do and *Prompt* is an optional message to the user what information to enter. The prompt can contain multiple words and should not be enclosed in quotation marks because they would end up showing in the final prompt.

Querying the user is just one of the many functions of Batcmd. It also has the ability to query all parts of the computer and return information via the errorlevel. Using Batcmd, your computer can find out things like how much memory is installed and available, how many drives are available, how much space on a drive is free, if a mouse is installed, what the date is, and much more. Only the Batcmd tools for querying the user are described in this chapter. The remainder of Batcmd is described in Chapter 25.

Single-character response questions

Most of the time, when you want to ask the user a question, you want to ask questions like "Do you really want to do this?" or "Which one of these menu items do you pick?" or even "Do you want to quit now?" Almost every time you ask the user a question, you will find that a single-character response is all you need. The errorlevel makes a perfect vehicle for communicating that information from the program to the batch file.

Batcmd offers several different methods of asking the user a question, depending on what type of information you need.

YN (Yes No)

This commands accepts only an "n" or "y" keystroke in either upper- or lowercase. For an "n" in either case, `Batcmd YN` sets errorlevel to 0; for a "y" in either case, it sets them to 1.

This is probably the Batcmd option you will use the most. Many of the questions you will want to ask the user are things like "Do you really want to erase all these files? (Y/N)" or "This takes six hours, do you want to continue? (Y/N)"; and using the YN option is so much nicer than using a Pause command and telling the user to press Ctrl-Break to stop the batch file.

GN (Get Number)

This command accepts any single-digit number from the user and exits with the errorlevel set to that number. If any other keystroke is entered, Batcmd beeps and continues waiting for a number. This command can display an optional prompt if it is included after the command.

When you need for the user to select between just a few options, this might be the best approach. If you need fewer than ten, just start your options at 0 and go back through the prompting loop if the user enters a number that is too high. You can do this with just one If-test. For example, if your last option is number 6, then loop back through the options anytime errorlevel is 7 or higher.

GL (Get Letter)

This command accepts any letter from the user and exits with the errorlevel set to 1 for an A, 2 for a B, and so on. If any other keystroke is entered, Batcmd beeps and

continues waiting for a letter. This command can display an optional prompt if it is included after the command.

If you are designing a menu system where you want the user to be able to make a selection using any letter, this option is the best way to implement it. It's also the best approach when you have a lot of options you are willing to letter sequentially. Because the errorlevels run sequentially from A, if your options stop at N (errorlevel 14), then you can test for an invalid response with a single If-test that tests for an errorlevel of 15 or higher.

However, if you want the user to use only a few of the letters or you want to pick-and-choose certain letters to use, then using the GF (get from list) option is a better approach than using this option and jumping back through the loop for invalid options.

GU (Get Uppercase Letter)

This command accepts only uppercase letters from the user. If the user enters a lowercase letter or any non-letter keystroke, Batcmd beeps and continues waiting for an uppercase letter. Then errorlevel is set to the ASCII value of the uppercase letter, so A is reported as 65, B is reported as 66, and so on. This command can display an optional prompt if it is included after the command. This is a specialized version of the GL (get letter) option. For most applications, the GL option is a more useful option.

GF (Get From List)

This command accepts a keystroke from the user only when that keystroke is on the list of acceptable keystrokes. The format of the command is `BATCMD GF` *`List Prompt`*, where *`List`* is a listing of all valid keystrokes without spaces between them and *`Prompt`* is the user message to display. The list of keystrokes is case-sensitive, so in order to accept only A through C from the user in either case, the command would have to be entered as `BATCMD GF AaBbCc` *`Prompt`*.

If the user enters an invalid keystroke, Batcmd beeps and continues waiting for a valid keystroke. Once the user enters a valid keystroke, Batcmd sets errorlevel to that keystroke's position in the list. So, for the above example, if the user entered B, the errorlevel would be set to 3 because the capital B is the third character in the list.

Batcmd checks the list from right to left. That is only an issue if you use duplicates in the list. If your list was Ronny, then n would be reported via an errorlevel of 4 and not 3 because the n in the fourth position was the first lowercase n encountered moving right to left.

This approach is designed for those occasions when you have a number of different selections for the user to pick from. If you have fewer selections, then having the user pick a number would be a better choice. At first glance, you might think that treating upper- and lowercase letters separately would double the number of If-tests in the batch file. That is not the case. Using the `BATCMD GF AaBbCc` *`Prompt`* shown earlier, we know errorlevel will be 1–6 and we want to treat upper- and lowercase the same. The following batch file segment does that:

```
BATCMD GF AaBbCc
IF ERRORLEVEL 5 GOTO C
IF ERRORLEVEL 3 GOTO B
IF ERRORLEVEL 1 GOTO A
```

We want to jump to the C section when errorlevel is 5–6; and because the IF ERRORLEVEL 5 statement is a greater-than-or-equal test and no values exceed 6, it handles both cases automatically. Once 5–6 have been eliminated, the IF ERRORLEVEL 3 statement tests for 3–4, which is B. The last statement could be replaced with just a GOTO A because everything else has been eliminated; I used an IF-test just for consistency's sake.

Of course, you can test for each capitalization separately if you need to. The following example illustrates that:

```
BATCMD GF AaBbCc
IF ERRORLEVEL 6 GOTO LITTLEC
IF ERRORLEVEL 5 GOTO BIGC
IF ERRORLEVEL 4 GOTO LITTLEB
IF ERRORLEVEL 3 GOTO BIGB
IF ERRORLEVEL 2 GOTO LITTLEA
IF ERRORLEVEL 1 GOTO BIGA
```

GK (Get Key) This command accepts any keystroke from the user and exits with the errorlevel set to the ASCII value of that keystroke. This command can display an optional prompt if it is included after the GK command.

Only very rarely will you need this tool. When you need to restrict entries, the GF option works better. The GF option also works better if you have less than 26 options because you can easily list all the letters. At 26 options, the GL is better because it handles getting just letters automatically. About the only time you will need this option is when you are doing something where you need to accept absolutely any keystroke from the user.

Multiple-character response questions

Occasionally, you want to ask the user a question that requires a multi-character response. Typically, this is used to ask the user for his name to customize the prompts or to ask for a specific filename. Multi-character responses cannot be stored in errorlevel because that variable only has room for one byte of information.

Batcmd has a way around this. The GE option asks the user a question and stores the results in the master copy of the environment under the name Batcmd. This command will display a prompt if one is entered on the command line after the GE.

Normally, a program like Batcmd does not have access to the master copy of the environment; if Batcmd were to just place the results in its copy of the environment, that information would be lost as soon as Batcmd terminated. That is clearly not what we want. Batcmd avoids the problem by searching through all the computer memory to find the master copy of the environment. It then makes its changes there.

Before using Batcmd, you might want to check the environment to see how much space is free; Batcmd has a CE option that does this for you. It sets the errorlevel to the number of free bytes in the environment up to 255. If more than 255 bytes are free, it sets errorlevel to 255. Users can enter up to 65 bytes of information in response to the GE prompt, and you also need room for the environmental variable name and some internal markers DOS uses. Thus, to be absolutely certain of not running out of space, you would want to check for a value of around 73. However, most users enter far less than 65 characters, so I usually tailor my tests to the type of information I'm asking for.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: SHOWGETS.BAT`
`REM PURPOSE: Show Using Batcmd In Bat`
`REM VERSION: 1.00`
`REM DATE: May 18, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`CLS`	Clear the screen.
`ECHO This Demonstration Takes About` `ECHO Ten Minutes To Run`	Explain what is happening to the user.
`BATCMD SL`	Use Batcmd to skip a line.
`BATCMD YN Do You Want To Continue? (Y/N)`	Use the yes/no option of Batcmd to ask the user a question.
`IF NOT ERRORLEVEL 1 GOTO END`	If the user did not answer yes, jump to the end of the batch file.
`ECHO I'm Glad You Decided To Stay`	Acknowledge that the user answered yes.
`BATCMD CE`	Use Batcmd to check and see how much environmental space is free.
`IF NOT ERRORLEVEL 25 GOTO ERROR`	If there is not 25-bytes of free environmental space, jump to an error-handling section.
`BATCMD SL`	Use Batcmd to skip a line.
`BATCMD GE Please Enter Your Name:`	Use Batcmd to ask the user's name. This is stored in the environment under the name Batcmd.
`CLS`	Clear the screen.
`GOTO TOP1`	Jump to the next section of the batch file.

19-2
SHOWGETS.BAT illustrates both using Batcmd to obtain information from the user and then acting on that information.

19-2
Continued.

Batch File Line	Explanation
:ERROR ECHO Your Environment Does Not Have Enough Free ECHO Space For The Entire Demonstration So I ECHO Am Skipping Asking Your Name ECHO In The Prompts, You Will See A Blank ECHO Where You Should See A Name PAUSE CLS GOTO TOP1	Section to explain the problem of inadequate free environmental space to the user.
:TOP1	Label marking the top of the first loop.
BATCMD SL	Use Batcmd to skip a line.
ECHO %BATCMD%:	Display the user's name if it was stored in the environment.
ECHO For The Next Question, Only A ECHO Number Will Be Acceptable, But ECHO You Might Want To Try Other ECHO Things First To See What Happens ECHO This Will Be Repeated Until ECHO You Answer With A 0	Tell the user what will happen next.
BATCMD GN Enter A Number 0-9	Ask the user to enter a number 0-9 using Batcmd.
IF ERRORLEVEL 9 ECHO 9 Entered IF ERRORLEVEL 8 IF NOT ERRORLEVEL 9 ECHO 8 Entered IF ERRORLEVEL 7 IF NOT ERRORLEVEL 8 ECHO 7 Entered IF ERRORLEVEL 6 IF NOT ERRORLEVEL 7 ECHO 6 Entered IF ERRORLEVEL 5 IF NOT ERRORLEVEL 6 ECHO 5 Entered IF ERRORLEVEL 4 IF NOT ERRORLEVEL 5 ECHO 4 Entered IF ERRORLEVEL 3 IF NOT ERRORLEVEL 4 ECHO 3 Entered IF ERRORLEVEL 2 IF NOT ERRORLEVEL 3 ECHO 2 Entered IF ERRORLEVEL 1 IF NOT ERRORLEVEL 2 ECHO 1 Entered IF ERRORLEVEL 8 IF NOT ERRORLEVEL 1 ECHO 0 Entered	Test the errorlevels in order to tell the user what number was selected.
IF ERRORLEVEL 1 GOTO TOP1	If the user entered anything other than zero, continue looping.
CLS	Clear the screen.
:TOP2	Label marking the top of the second loop.
ECHO %BATCMD%:	Display the user's name if it was stored in the environment.
ECHO For The Next Demonstration Only ECHO A Letter Will Be Accepted ECHO You Might Want To Try Other Things First ECHO The Demonstration Will Continue Until ECHO You Enter An A	Tell the user what will happen next.
BATCMD SL	Use Batcmd to skip a line.

19-2
Continued.

Batch File Line	Explanation
`BATCMD GL Enter Any Letter`	Use Batcmd to ask the user a question and accept any letter in response.

`IF ERRORLEVEL 1 IF NOT ERRORLEVEL 2 ECHO A Entered`
`IF ERRORLEVEL 2 IF NOT ERRORLEVEL 3 ECHO B Entered`
`IF ERRORLEVEL 3 IF NOT ERRORLEVEL 4 ECHO C Entered`
`IF ERRORLEVEL 4 IF NOT ERRORLEVEL 5 ECHO D Entered`
`IF ERRORLEVEL 5 IF NOT ERRORLEVEL 6 ECHO E Entered`
`IF ERRORLEVEL 6 IF NOT ERRORLEVEL 7 ECHO F Entered`
`IF ERRORLEVEL 7 IF NOT ERRORLEVEL 8 ECHO G Entered`
`IF ERRORLEVEL 8 IF NOT ERRORLEVEL 9 ECHO H Entered`
`IF ERRORLEVEL 9 IF NOT ERRORLEVEL 10 ECHO I Entered`
`IF ERRORLEVEL 10 IF NOT ERRORLEVEL 11 ECHO J Entered`
`IF ERRORLEVEL 11 IF NOT ERRORLEVEL 12 ECHO K Entered`
`IF ERRORLEVEL 12 IF NOT ERRORLEVEL 13 ECHO L Entered`
`IF ERRORLEVEL 13 IF NOT ERRORLEVEL 14 ECHO M Entered`
`IF ERRORLEVEL 14 IF NOT ERRORLEVEL 15 ECHO N Entered`
`IF ERRORLEVEL 15 IF NOT ERRORLEVEL 16 ECHO O Entered`
`IF ERRORLEVEL 16 IF NOT ERRORLEVEL 17 ECHO P Entered`
`IF ERRORLEVEL 17 IF NOT ERRORLEVEL 18 ECHO Q Entered`
`IF ERRORLEVEL 18 IF NOT ERRORLEVEL 19 ECHO R Entered`
`IF ERRORLEVEL 19 IF NOT ERRORLEVEL 20 ECHO S Entered`
`IF ERRORLEVEL 20 IF NOT ERRORLEVEL 21 ECHO T Entered`
`IF ERRORLEVEL 21 IF NOT ERRORLEVEL 22 ECHO U Entered`
`IF ERRORLEVEL 22 IF NOT ERRORLEVEL 23 ECHO V Entered`
`IF ERRORLEVEL 23 IF NOT ERRORLEVEL 24 ECHO W Entered`
`IF ERRORLEVEL 24 IF NOT ERRORLEVEL 25 ECHO X Entered`
`IF ERRORLEVEL 25 IF NOT ERRORLEVEL 26 ECHO Y Entered`
`IF ERRORLEVEL 26 IF NOT ERRORLEVEL 27 ECHO Z Entered`
	Test the errorlevels in order to tell the user what letter was selected.

`IF ERRORLEVEL 2 GOTO TOP2`	If the user entered anything other than the letter A, continue looping.
`CLS`	Clear the screen.
`:TOP3`	Label marking the top of the third loop.
`ECHO %BATCMD%:`	Display the user's name if it was stored in the environment.

`ECHO For The Next Demonstration Only`
`ECHO An Uppercase Letter Will Be Accepted`
`ECHO You Might Want To Try Other Things First`
`ECHO The Demonstration Will Continue Until`
`ECHO You Enter An A`
	Tell the user what will happen next.

`BATCMD SL`	Use Batcmd to skip a line.
`BATCMD GU Enter Any UPPERCASE Letter`	Use Batcmd to ask the user a question and accept any uppercase letter in response.

Batch File Line	Explanation
`ECHO %BATCMD%:`	Display the user's name if it was stored in the environment.

`ECHO For The Next Demonstration`
`ECHO An Uppercase Letter Will Be Accepted`
`ECHO You Might Want To Try Other Things First`
`ECHO The Demonstration Will Continue Until`
`ECHO You Enter An A`
	Tell the user what will happen next.

`BATCMD SL`	Use Batcmd to skip a line.
`BATCMD GU Enter Any UPPERCASE Letter`	Use Batcmd to ask the user a question and accept any uppercase letter in response.

`IF ERRORLEVEL 65 IF NOT ERRORLEVEL 66 ECHO A Entered`
`IF ERRORLEVEL 66 IF NOT ERRORLEVEL 67 ECHO B Entered`
`IF ERRORLEVEL 67 IF NOT ERRORLEVEL 68 ECHO C Entered`
`IF ERRORLEVEL 68 IF NOT ERRORLEVEL 69 ECHO D Entered`
`IF ERRORLEVEL 69 IF NOT ERRORLEVEL 70 ECHO E Entered`
`IF ERRORLEVEL 70 IF NOT ERRORLEVEL 71 ECHO F Entered`
`IF ERRORLEVEL 71 IF NOT ERRORLEVEL 72 ECHO G Entered`
`IF ERRORLEVEL 72 IF NOT ERRORLEVEL 73 ECHO H Entered`
`IF ERRORLEVEL 73 IF NOT ERRORLEVEL 74 ECHO I Entered`
`IF ERRORLEVEL 74 IF NOT ERRORLEVEL 75 ECHO J Entered`
`IF ERRORLEVEL 75 IF NOT ERRORLEVEL 76 ECHO K Entered`
`IF ERRORLEVEL 76 IF NOT ERRORLEVEL 77 ECHO L Entered`
`IF ERRORLEVEL 77 IF NOT ERRORLEVEL 78 ECHO M Entered`
`IF ERRORLEVEL 78 IF NOT ERRORLEVEL 79 ECHO N Entered`
`IF ERRORLEVEL 79 IF NOT ERRORLEVEL 80 ECHO O Entered`
`IF ERRORLEVEL 80 IF NOT ERRORLEVEL 81 ECHO P Entered`
`IF ERRORLEVEL 81 IF NOT ERRORLEVEL 82 ECHO Q Entered`
`IF ERRORLEVEL 82 IF NOT ERRORLEVEL 83 ECHO R Entered`
`IF ERRORLEVEL 83 IF NOT ERRORLEVEL 84 ECHO S Entered`
`IF ERRORLEVEL 84 IF NOT ERRORLEVEL 85 ECHO T Entered`
`IF ERRORLEVEL 85 IF NOT ERRORLEVEL 86 ECHO U Entered`
`IF ERRORLEVEL 86 IF NOT ERRORLEVEL 87 ECHO V Entered`
`IF ERRORLEVEL 87 IF NOT ERRORLEVEL 88 ECHO W Entered`
`IF ERRORLEVEL 88 IF NOT ERRORLEVEL 89 ECHO X Entered`
`IF ERRORLEVEL 89 IF NOT ERRORLEVEL 90 ECHO Y Entered`
`IF ERRORLEVEL 90 IF NOT ERRORLEVEL 91 ECHO Z Entered`
	Test the errorlevels in order to tell the user what letter was selected.

`IF ERRORLEVEL 66 GOTO TOP3`	If the user entered anything other than the letter A, continue looping.
`CLS`	Clear the screen.
`:TOP4`	Label marking the top of the fourth loop.
`ECHO %BATCMD%:`	Display the user's name if it was stored in the environment.

`ECHO For This Next Demonstration`
`ECHO Only The Letters "Ronny" Will Be Accepted`
`ECHO And The Capitalization Must Match Exactly`
`ECHO You Might Want To Try Other Entries First`
	Tell the user what will happen next.

`BATCMD SL`	Use Batcmd to skip a line.
`BATCMD GF Ronny Select From The Letters "Ronny"`	Use Batcmd to ask the user a question and accept only letters in the list.

19-2
Continued.

Batch File Line	Explanation
IF ERRORLEVEL 1 IF NOT ERRORLEVEL 2 ECHO R Entered IF ERRORLEVEL 2 IF NOT ERRORLEVEL 3 ECHO o Entered IF ERRORLEVEL 4 IF NOT ERRORLEVEL 5 ECHO n Entered IF ERRORLEVEL 5 IF NOT ERRORLEVEL 6 ECHO y Entered	Test the errorlevels in order to tell the user what letter was selected.
IF ERRORLEVEL 2 GOTO TOP4	If the user entered anything other than the letter R, continue looping.
CLS	Clear the screen.
ECHO %BATCMD%:	Display the user's name if it was stored in the environment.
ECHO For This Next Demonstration Any Character ECHO Will Be Accepted And Its Errorlevel Shown	Tell the user what will happen next.
BATCMD GK Enter Any Keystroke	Use Batcmd to ask the user a question and accept any keystroke.
CALL SAYERROR	Run the SAY-ERROR.BAT batch file to display the errorlevel.
:END	Label marking the end of the batch file.

19-2
Continued.

Which of these many Batcmd options you use to query users really depends on the type of information you are asking for. To give you a better idea of how to use each option, SHOWGETS.BAT in FIG. 19-2 uses each of these options in a functioning batch file. Most of the examples perform logic testing on the user's response. SHOWGETS.BAT requires the Batcmd program and the SAYERROR.BAT batch file to either be in the current subdirectory or in the path when it runs.

Summary

In DOS 6, the CHOICE command gives you a way of getting information from the user and placing it in the errorlevel. CHOICE accepts a prompt and list of acceptable characters and when run requires the user to enter one of these acceptable characters. Unless a special switch is used, CHOICE is case-insensitive. There are also switches to hide the list of acceptable characters and to cause CHOICE to make a default selection after 0–99 seconds.

The YN option of Batcmd only accepts a y or n keystroke and reports the results via the errorlevel. This is perfect for yes/no questions.

The GN option of Batcmd only accepts numbers 0–9 and reports the results via the errorlevel. This option is perfect for menus or questions needing ten or fewer responses.

The GL option of Batcmd only accepts any letter without regard for case and reports the results via the errorlevel. The letter A is reported as 1, B as 2, and so on. This option is useful for menus or questions needing more than ten responses.

The GU option of Batcmd is a specialized version of the GL options that only accepts uppercase letters. The results are reported via errorlevel using the ASCII value of the letter.

The GF option of Batcmd only accepts keystrokes that are part of a list supplied to the GF command. The results are reported via errorlevel using the position of keystroke in the list of valid keystrokes. This option is useful for selecting from specialized lists.

The GK option of Batcmd accepts any keystroke and reports in ASCII code via the errorlevel.

The GE option of Batcmd accepts a multi-character response and stores that response in the environment under the name Batcmd.

The CE option of Batcmd reports on the amount of free environmental space and sets errorlevel to the number of bytes free up to 255. Over 255, the errorlevel is set to 255.

20 Advanced screen design 1

The main tool for displaying messages in a batch file is a series of ECHO commands, which works well for a few messages but soon gets old if your batch file has to display a lot of information. Fortunately, you have several alternatives. You can

- create a text file and use the TYPE command to send that file to the screen.
- write a custom program.
- use a screen compiler.
- use ANSI.

The alternative you select depends on your specific needs.

Text files

Probably the easiest way to quickly display a screen full of information is to use the DOS editor or your word processor to create each screen as a separate ASCII file. That way, you can add blank lines where you want them, center text, and otherwise visually arrange the elements on the screen. Once you have the screens looking the way you want, just type them to the screen from your batch file. Of course, you'll need a PAUSE command between each screen.

Using this method, a batch file might look like this:

```
@ECHO OFF
TYPE C:\TEXT\INTRO.TXT
PAUSE
TYPE C:\TEXT\SCREEN1.TXT
PAUSE
TYPE C:\TEXT\SCREEN2.TXT
PAUSE
```

Notice that this batch file segment specifies the full path to each text file. Because batch files are usually designed to run from anywhere on your hard disk, you cannot depend on the batch file and the associated text files being in the same subdirectory from which the batch file is executed.

This method is easier than a series of ECHO commands but has several drawbacks. One such drawback is simply remembering to send copies of all the associated text files when you send someone a copy of the batch file. A second drawback is that anyone wanting to run your batch file on their system must have a subdirectory of the same name you used to store your text files and then must copy the text files to that subdirectory. That's a lot to expect users to do. (Of course, you could write a batch file to install this batch file on their system.) Finally, this method lacks the ability to control the screen color.

Remember: Typing an ASCII file to the screen is an easy way for a batch file to display a screen full of information, but this method creates unique problems when batch files are to be distributed to other users. You must remember to give them all the files the batch file uses and they must store them in the right subdirectory.

Writing a program

You can also display text by using a high-level programming language. Writing high-level programs is beyond the scope of this book, however, so I won't cover it in any detail. Most high-level programs compile to an executable .EXE file, so all you must do to display the text is include the name of the .EXE file in a batch file. In addition, most high-level languages can control the screen color and execute very quickly.

This approach allows you to avoid the problem of needing a specific directory name to store the file in. As long as the resulting screen is anywhere in the path, it will execute whenever the batch file issues the command to run the program. However, while we've solved one problem, we've introduced another problem—namely the programming tools and abilities this approach requires—one thing many readers lack.

Remember: Writing a program is a nice way to create custom screen, but it also requires skills and tools that many readers do not have.

Using a screen compiler

What would be nice would be having a method of creating screen display programs as nice as a programming language allows without the need to buy a programming language and learn to use it. Well, the disk that comes with this book has just that! It contains a screen compiler called BatScreen. BatScreen was written especially for this book by Doug Amaral of hyperkinetix, the maker of the Builder batch file compiler. The BatScreen screen compiler combines the ease of creating an ASCII file with the power of writing a program.

BatScreen takes an ASCII text file and converts it to a small .COM file. When you enter the name of the .COM file at the command line prompt or in a batch file, the file will flash up on the screen almost instantaneously.

To run BatScreen and compile a screen, first create the screen with any ASCII editor. Arrange the text in the file the way you want it to appear on the screen. Feel free to use high-ordered ASCII characters. Once the screen is ready, change to the subdirectory containing the ASCII file and enter `BS` at the command line prompt. BatScreen first presents a screen showing all the non-blinking color choices with a box around the currently selected color combination. Use the cursor to move the

box to where it surrounds the color combination you want. If you want the text to blink, press PgDn and select from the blinking text in the same fashion.

Next, BatScreen prompts you for the name of an ASCII file. To completely fill the screen, the ASCII file should contain 80 columns and 24 rows. If the file is larger, BatScreen will ignore the extra. Finally, BatScreen will ask you if you want to clear the screen when the program displays. Answer "Yes" and the .COM file will clear the entire screen, set it to the colors you selected, and display the contents of the ASCII file. Answer "No" and the .COM file will clear off only enough lines to display the message. Blinking text and not clearing the screen is perfect for error messages and other occasions where you want to grab the user's attention. Full screen solid colors are great for menus and general-purpose text.

The original ASCII file isn't modified and doesn't need to be present for the .COM file to operate, so you can modify and recompile it if you ever need to change the screen. Additionally, once you've created the .COM file, BatScreen is not needed to display the screen.

Remember: The BatScreen screen compiler is probably the best way to create custom screens.

BatScreen makes excellent menus. It also has several other useful functions:

- Several of the programs I use don't properly reset the cursor when they terminate. I end up with a flashing box or some other strange shape for a cursor. I've compiled an ASCII file containing a single blank space; running the resulting .COM file will reset the screen colors and sometimes resets the cursor to its proper shape.
- I like my display to have white letters and a blue background. This is initially set by the BatScreen menu that my AUTOEXEC.BAT file displays. However, these colors are easily reset to white on black by a CLS (clear screen) command or by a number of application programs that don't reset screen color when they terminate. I have a BatScreen file called COLOR.COM that was compiled from an ASCII file consisting of a single space. When I issue the Color command, the screen is cleared and my colors reset.
- Many times I need to display a message from within a batch file. By compiling them with BatScreen and using blinking colors, I can generate messages that flash up in unexpected colors. These messages get noticed!

After you try out BatScreen, I'm sure you'll agree that Doug has produced an excellent tool for batch file menus and screens. In fact, I use it almost exclusively for my screens.

Ansi

Because Ansi is such a large topic, I've devoted all of Chapter 21 to it. At the moment, I'll just say that Ansi allows you to control almost every aspect of the appearance of the screen but also requires loading the device driver ANSI.SYS—something most users do not do.

Summary

You can display a screen full of information with a series of ECHO commands.

Another way to display a screen full of information is by creating an ASCII file and typing it to the screen.

Using an ASCII file to display text when the batch file is shared with other users requires sharing the text files as well and requires the users to install the text files in the same subdirectory as was used on the original system.

Another way to display a screen full of information is by writing your own program. This requires programming tools and experience with a programming language.

If you use a program to display a screen full of information, the program must be shared with other users of the batch file; it can still be placed in any subdirectory along the path, though.

Another way to display a screen full of information is to create an ASCII file and compile it with the BatScreen screen compiler.

When a screen program is created with BatScreen and the batch file is shared with other users, the screen program must be shared as well and must be installed in a subdirectory along the path.

The final way to display a screen full of information is by using ANSI commands. ANSI is covered in Chapter 21.

21 Advanced screen design 2: ANSI.SYS

Your main tool for displaying messages in a batch file is a series of ECHO commands. That works well for a few messages but soon gets old if your batch file must display a lot of information. In the last chapter, we examined using text files and the TYPE command, writing custom programs, and using a screen compiler to design screens. A fourth method of screen design is Ansi. Because Ansi is such a complex program, I decided to cover it here in a separate chapter instead of including it with the other methods.

Introduction

The device driver ANSI.SYS (Ansi for short) gives you an incredible amount of power over the appearance of the screen. By using that power, you can produce very attractive and eye-catching screens and messages in your batch files. However, writing batch files for others using Ansi is asking for problems because so few users load Ansi. Nothing looks worse than seeing Ansi escape sequences stream across a screen when Ansi is not running. The reason for Ansi's lack of popularity is that it's saddled with the worst user interface of any program on the market! Ansi makes Edlin look user-friendly!

Warning: Ansi is a great way to create colorful screens but only if everyone who runs the resulting batch file will be loading Ansi.

Getting started with Ansi

To use Ansi, you must first load it in your CONFIG.SYS file with a statement like

```
DEVICE=C:\DOS\ANSI.SYS
```

Of course, be sure to use the subdirectory where you have Ansi stored. This is generally your DOS subdirectory. Ansi will use around 4K of memory, which is actually very little memory to sacrifice for the power that Ansi gives you.

Before reading about Ansi, you might want to try a little experiment. Configure your computer to load Ansi and reboot. Once you have done that, run the ANSIDEMO.BAT batch file (shown in FIG. 21-1).

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: ANSIDEMO.BAT`
`REM PURPOSE: Show ANSI.SYS Power`
`REM To Control The Screen`
`REM VERSION: 2.00`
`REM DATE: July 22, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`REM Change The Screen Colors` `REM Bright White On Blue`	Documentation remarks.
`ECHO ESC[37m`	Send an ANSI sequence to set the foreground color to white. (Note: the batch file uses an ASCII-27 escape character, not ESC.)
`ECHO ESC[44m`	Send an ANSI sequence to set the background color to blue.
`ECHO ESC[1m`	Send an ANSI sequence to set the foreground color attribute to bright.

`REM Clear the Screen`
`REM Set Colors First So CLS`
`REM Would Use New Colors`
	Documentation remarks.

`ECHO ESC[2J`	Send an ANSI sequence to clear the screen.
`REM Change To Flashing Red` `REM Display Message And Reset`	Documentation remarks.
`ECHO ESC[1;30HESC[31mESC[5mA Very Special MenuESC[0mESC[37mESC[44mESC[1m`	Send the following ANSI sequences: ESC[1;30H to position the cursor on row one and column thirty. ESC[31m to set the foreground color to red. This affects only the text that is written to the screen after this. ESC[5A to set the foreground text to blinking. At this point, a message is written to the screen. ESC[0m to reset the screen display to white on black. This affects only future text and is used here to stop the blinking. ESC[37mESC[44m ESC[1m to set the display to bright white on blue.
`REM Position Cursor & Display Options`	Documentation remark.

21-1
ANSIDEMO.BAT demonstrates some of the formatting power of ANSI.SYS. Before running ANSIDEMO.BAT, you must load ANSI.SYS in your CONFIG.SYS file and reboot.

Batch File Line	Explanation
`ECHO ESC[6;20H1.  Run Word Processing`	Position the cursor on row six at column twenty and write text.

`ECHO ESC[7;20H2.  Run Spreadsheet`
`ECHO ESC[8;20H3.  Run Database`
`ECHO ESC[9;20H4.  Play Games`
`ECHO ESC[10;20H5.  Format A Disk`
`ECHO ESC[11;20H6.  Backup Hard Disk To Floppies`
	Write the remaining menu options to the screen.

`ECHO ESC[15;20HESC[33mPress The Number of The Program You WantESC[37m`	Position the cursor on row fifteen and column twenty, change the foreground color to yellow, write text to the screen, and reset the foreground color.

`ECHO`
`PAUSE`
`GOTO END`
	Pause until the user presses a key and then exit the batch file.

`:HELP`
`ECHO This Batch File Is A`
`ECHO Demonstration Of ANSI`
`ECHO It Requires That You Load`
`ECHO ANSI.SYS In Your CONFIG.SYS File`
`GOTO END`
	Section that displays help when the user starts the batch file with a /? or a ? as the first replaceable parameter.

`:END`	Label marking the end of the batch file.

21-1
Continued.

If you have a color display, ANSIDEMO.BAT gave you a blue background with bright white letters for the text. There was a flashing red title at the top of the menu and a bright yellow message at the bottom saying to make your selection. Now, to see the real power of Ansi, issue a CLS command.

With other utilities that set the color of the display, CLS returns the display to white characters on a black background. However, Ansi color assignments are permanent, at least until you reboot or issue a new Ansi command, so the CLS command retains the blue background with bright white lettering. Because ANSIDEMO.BAT used nothing but DOS and Ansi commands, this batch file will run on any computer that loads Ansi. As a result, you don't have to worry about the user owning a utility program to be able to change colors or position the cursor. You also do not have to worry about any supporting files.

If you like what you saw, read on: What follows is a tutorial on Ansi. Ansi has fifteen separate functions which fall into four broad categories:

- **Screen control.** You have already seen this in the example. Ansi controls screen colors nicely and allows you to display different text on the screen in different colors. It also allows you to change your prompt to prompts far fancier than the pitiful (by comparison) prompts that DOS allows without Ansi.
- **Cursor control.** Ansi allows you to position the cursor anywhere on the screen. ANSIDEMO.BAT used this to center text and otherwise control its position on the screen without the need to pad with spaces or issue a specific number of Echo commands to move the cursor down the screen.

- ❑ **Video control.** Ansi allows you to easily change the display mode and attributes.
- ❑ **Keyboard control.** Ansi allows you to remap your keyboard and write keyboard macros.

First, a warning: Ansi is very difficult to learn to use, due to the poor explanation in the manual plus its unusual user interface. Plan on spending a lot of time studying this tutorial and your DOS manual—along with doing a lot of experimenting—before you can consider yourself an Ansi master.

Warning: While very powerful, Ansi has a very poor interface that is difficult to learn and hard to use.

Entering Ansi commands

Every Ansi command starts with an escape followed by a left square bracket. The Ansi command to clear the screen, for example, is ESC[2J. (For all the commands and batch file listings that follow, I have used ESC to stand for the escape character so you can read it, both in the text and the batch file listings. However, the batch file or command must have the escape character, which has an ASCII value of 27 and a hexadecimal value of 1B.) Because the first character in an Ansi command is always an escape, the commands are often called *escape sequences*.

Remember: Ansi commands are called escape sequences.

From this description so far, you should immediately see the first problem with Ansi: you cannot enter any Ansi commands from the keyboard because when you press the Esc key, DOS thinks you want to abort the current command. Typing Alt-27 on the number pad does not bypass this problem; it's just another way of entering an escape. So that you can experiment, I have included a batch file called SENDANSI.BAT on the disk that comes with this book. It has command line help and remarks, so it's fairly long, but the only command it uses is `ECHO ESC[%1`. To use it, you enter the Ansi command you want to run without the escape and left bracket that begins the command. SENDANSI.BAT adds those two characters and issues the command. SENDANSI.BAT will not work with Ansi keyboard reassignment commands or any other Ansi commands that include a semicolon or require more than one "word" of commands.

Word processors, editors, & "escaping" problems

Many word processors and editors make it difficult to create batch files with Ansi commands because they refuse to let you enter an escape character into the file. There is a trick to get around that. When you enter the CLS command, DOS responds with an ESC[2J command, even if Ansi is not loaded. You can redirect that to a file with the command `CLS>file`. Now, you can edit this file with your word processor. While many word processors do not allow you to enter an escape, most will edit a file containing an escape without any problem. You just copy the escape character to your scrap and paste it anywhere you need it.

Remember: Redirect a CLS command to a file to easily get an escape into a file.

The DOS 5 and 6 editor lets you enter the escape directly. Press Ctrl-P and then Esc. You will see a left-pointing arrow on the screen. That is the escape character. Now, just enter the rest of the Ansi escape sequence.

Remember: With the DOS editor, press Ctrl-P followed by Esc to enter an escape character into a file.

☑

Sending Ansi commands by typing a file

While Ansi is loaded, all the text that is written to the screen through DOS is processed through Ansi. Ansi watches for the ESC[sequence. Anything beginning with ESC[is treated as an Ansi command and anything else is sent to the screen. That gives us a number of ways to send Ansi sequences to the screen. To see that, issue the following two commands:

```
CLS > TEMP.BAT
TYPE TEMP.BAT
```

If you look at the file TEMP.BAT with an editor, you will see it contains the text sequence "ESC[2J." However, when you try to display it with the TYPE command, Ansi sees the sequence beginning with ESC[as a command and executes it. So, in addition to echoing Ansi commands to the screen with the ECHO command, you can put them in an ASCII file and type them to the screen. You can also copy these ASCII files to the screen with the command:

```
COPY TEMP.BAT CON
```

Remember: You can give Ansi commands by placing the escape sequences in an ASCII file and typing or copying that file to the screen.

☑

Sending Ansi commands using the environment

Now, to continue with the experiment, load TEMP.BAT into an editor. Add the command `SET TEMP=` in front of the Ansi sequence so the line looks like `SET TEMP=ESC[2J`. Now, exit and run the batch file. Because you did not turn command echoing off, DOS tries to echo the command to the screen, and again Ansi intercepts it and executes it. However, the environmental variable was successfully created.

Now, enter the SET command. DOS displays the other environmental variables properly. However, when it reaches the TEMP variable and tries to display its contents, Ansi again intercepts and executes the escape sequence. In addition to ECHO, TYPE, and COPY, this gives us a fourth way to issue Ansi commands. It also turns out that this is the easiest way.

Remember: You can give Ansi commands by placing the escape sequences in an environmental variable and using the SET command to display the environmental variables.

☑

Sending Ansi commands via the prompt

Most environmental variables are rarely displayed. There is, however, one exception. The prompt is an environmental variable that is displayed after every DOS command. It is also displayed for each batch file command if command echoing is on. The prompt is particularly useful for Ansi commands. The metacharacter $e issues an escape, so you avoid the problem of having to get an escape into the sequence. In addition, the prompt command has other metacharacters to spice up a prompt. TABLE 21-1 lists them.

Table 21-1
The metacharacters you can include in a prompt statement. For ANSI, $e is very useful because it issues an escape symbol.

Command	Action
$$	Display a dollar sign.
$_	Include a carriage return and line feed.
$b	Display a vertical bar.
$d	Display the date.
$e	Include an escape. This is useful for sending Ansi escape sequences via the promt command.
$g	Display a greater-than sign.
$h	Display a backspace—thus deleting the prior character.
$l	Display a less-than sign.
$n	Display the current drive.
$p	Display the current subdirectory.
$q	Display an equal sign.
$t	Display the time.
$v	Display the DOS version.

☑ Remember: You can give Ansi commands by placing the escape sequences in the prompt. This is especially easy because the prompt offers the $e metacharacter that represents the escape character.

You can see an example of the fancy prompts Ansi is capable of generating by issuing the following command from the command line prompt with Ansi loaded:

```
PROMPT=$e[s$e[1;1H$e[K$d..$t..$p$e[u$p$g$e[44m$e[37m$e[1m
```

Replace the dots with an equal number of spaces. This prompt shows the day of the week, the date, the time, and the current drive and subdirectory on the top line of the screen. It also sets the screen colors to bright white on blue and uses the standard PG prompt on the current line of the screen!

This prompt uses a couple of Ansi sequences I have not yet mentioned, but you see how elements of the prompt metacharacters and Ansi can be combined to produce an exceptional prompt. In case you do not want to take the time to enter and debug this complex prompt, the batch file NICEPROM.BAT on the disk that comes with this book will issue it for you.

You should be aware of two problems with using the prompt to send Ansi escape sequences in a batch file. Normally, the first command of a batch file is `@ECHO OFF` to turn command-echoing off. However, this also causes DOS to stop displaying the prompt, which prevents Ansi escape sequences stored in the prompt from being sent to the screen so they can be processed. The solution is to issue an ECHO ON command when you want to send Ansi escape sequences with the prompt. After sending the Ansi escape sequence, you can turn echo back off so that your batch file will execute cleanly. ANSIHIDE.BAT in FIG. 21-2 illustrates this.

Warning: When using the PROMPT command in a batch file to send Ansi escape sequences, command echoing must be turned on. Otherwise, the prompt is never

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: ANSIHIDE.BAT`
`REM PURPOSE: Use ANSI.SYS To Hide Messages`
`REM VERSION: 1.00`
`REM DATE: December 19, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`ECHO ON`	Turn command-echoing back on. Since the Ansi escape sequences are being sent via the prompt, the prompt must be displayed for DOS to receive the escape sequences. For the prompt to be displayed, echo must be on.
`PROMPT=$e[0;40;37m`	Send the Ansi escape sequences to change the screen color to white-on-black.
`ECHO`	Display a blank line by echoing Alt-255. The real purpose is to display the prompt so DOS will receive the Ansi escape sequence.
`@ECHO OFF`	Turn command-echoing back off.
`CLS`	Clear the screen.
`ECHO Message 1 (Visible)` `ECHO Message 2 (Visible)`	Display two messages using DOS, the same method that the programs that come with DOS display their messages.
`ECHO ON`	Turn command echoing on in order to send another Ansi escape sequence to DOS.
`PROMPT $e[0;30;40m`	Send the Ansi escape sequence to change the screen color to black-on-black.
`ECHO`	Echo Alt-255 to force DOS to display the prompt so it can receive the Ansi escape sequences.
`@ECHO OFF`	Turn command-echoing back off.
`ECHO Message 3 (Invisible)` `ECHO Message 4 (Invisible)`	Send two more messages using DOS. Since these are displayed as black text on a black background, they are invisible.

`ECHO ON`
`PROMPT=$e[0;40;37m`
`ECHO`
`@ECHO OFF`
	Reset the screen colors to white-on-black.

`ECHO Message 5 (Visible)` `ECHO Message 6 (Visible)`	Display two more messages.
`PROMPT $p$g`	Reset the prompt.
`GOTO END`	Exit the batch file.

21-2

ANSIHIDE.BAT uses the prompt to send ANSI escape sequences. It turns command echoing on and off as needed. This batch file also shows how to hide DOS messages that are being sent to the screen.

21-2
Continued.

Batch File Line	Explanation
:HELP ECHO This Is A Demonstration ECHO Batch File That Uses ANSI ECHO To Hide Messages GOTO END	Section that displays help when the user starts the batch file with a /? or a ? as the first replaceable parameter.
:END	Label marking the end of the batch file.

Warning: When using the PROMPT command in a batch file to send Ansi escape sequences, command echoing must be turned on. Otherwise, the prompt is never displayed.

Second, prompts used to send Ansi escape sequences tend to be much longer than "standard" prompts. Because your prompt is stored in the environment, you must make sure you have enough free environmental space for these longer prompts.

I created a batch file called SENDANS2.BAT so that you could experiment with all the combinations of Ansi commands you like. To run it, you enter the command

```
SENDANS2 command1 command2 ...
```

at the command line. SENDANS2.BAT adds the ESC[to the beginning of each command, so you should leave it off of all commands. It builds a large environmental variable containing the entire command string and then issues it all at once, so there are no linefeeds between commands. However, that means your environment must be large enough to store all of the commands you plan on issuing, plus the ESC[that SENDANS2.BAT adds to the beginning of each command. Finally, because SENDANS2 strips off one character from each parameter (the leading space) and adds two (ESC[), the environmental variable will be longer than the command you issue. If that causes it to exceed the 127-character limit DOS places on commands, the computer could lock, reboot, or misbehave in some other unpredictable manner. If you keep your commands to 80 characters or less, this will not affect you.

Like SENDANSI.BAT, SENDANS2.BAT cannot be used to send Ansi commands containing a semicolon because DOS treats the semicolon as a divider and strips it off before passing the replaceable parameter to the batch file. As a result, there is no way to get the colon on the command line into the batch file.

Screen control

The most common Ansi command is the command to change colors. The command is ESC[#m, where # is the number of the foreground or background color to use. The color numbers are all two digits; all foreground colors start with a 3 and all background colors start with a 4. The colors are as follows:

0 Black
1 Red
2 Green
3 Yellow
4 Blue
5 Magenta
6 Cyan
7 White

So the command ESC[44mESC[37m sets the screen to white letters on a blue background—my personal favorite. TABLE 21-2 shows all of the associated Ansi escape sequences.

Table 21-2
Ansi color escape sequences.

Escape Sequence	Function
ESC[30m	Set foreground color to black.
ESC[31m	Set foreground color to red.
ESC[32m	Set foreground color to green.
ESC[33m	Set foreground color to yellow.
ESC[34m	Set foreground color to blue.
ESC[35m	Set foreground color to magenta.
ESC[36m	Set foreground color to cyan.
ESC[37m	Set foreground color to white.
ESC[40m	Set background color to black.
ESC[41m	Set background color to red.
ESC[42m	Set background color to green.
ESC[43m	Set background color to yellow.
ESC[44m	Set background color to blue.
ESC[45m	Set background color to magenta.
ESC[46m	Set background color to cyan.
ESC[47m	Set background color to white.

Three notes are in order here:

- These color numbers are not the same as the numbers you are used to using if you program in Basic.
- The clear screen command ended with an uppercase J, while the color command ended with a lowercase m. Most Ansi commands are case-sensitive, so pay attention to case while writing Ansi commands.
- An m ends every Ansi attribute setting command, but you only need one per line. Thus, the previous command could be shortened to ESC[44;37m. The m terminates the command. You do not have to reissue the ESC[after the "44" because the Ansi command is not terminated due to the fact that there is no m.

Being able to control the colors leads to two somewhat useful utilities. BLANK.BAT uses Ansi sequences to blank the screen at the command line prompt by converting the screen to black text on a black background. BLANK.BAT is shown below:

```
@ECHO OFF
ECHO ESC[0m
ECHO ESC[30m
ECHO ESC[40m
CLS
```

This keeps amateur snoops from looking at your files. I should note that the blank screen is really only blank at the command line and the few programs that use Ansi color settings. Most programs, however, display fine while Ansi is set for a blank

screen. Of course, the user must be able to load the program without seeing the prompt of the commands being entered. Typing UNBLANK runs UNBLANK.BAT, shown here:

```
@ECHO OFF
ECHO ESC[37m
ECHO ESC[44m
ECHO ESC[1m
CLS
```

UNBLANK.BAT uses Ansi commands to restore the screen to bright white letters on a blue background. (You must, of course, be able to run UNBLANK.BAT without being able to see the screen.) Note that the batch files on your disk have more documentation and command line help. However, that does not aid in the explanation here, so it is not reproduced here.

Ansi also provides for several types of screen erasing:

- `ESC[0J` Erases the screen from the cursor position to the end of the screen. (Note: Not all versions of Ansi support this command.)
- `ESC[0K` Erases the line the cursor is on, but only from the cursor position to the right.
- `ESC[1J` Erases the screen from the cursor position to the top of the screen. (Note: Not all versions of Ansi support this command.)
- `ESC[1K` Erases the line the cursor is on, from the beginning of the line to the current cursor position. (Note: Not all versions of Ansi support this command.)
- `ESC[2J` Clears the entire screen identically to a CLS command. In fact, the CLS command causes DOS to issue an ESC[2J command.
- `ESC[2K` Erases the entire line that the cursor is currently on. (Note: Not all versions of Ansi support this command.)

Cursor control

Ansi gives you the ability to position the cursor anywhere on the screen and to move the cursor around the screen as you see fit. In their simplest form, the commands to move around the screen are

- `ESC[A` Move the cursor up one row.
- `ESC[B` Move the cursor down one row.
- `ESC[C` Move the cursor to the right one column.
- `ESC[D` Move the cursor to the left one column.

If you precede the A through D with a number, the cursor is moved that many times, so `ESC[5A` moves the cursor up five rows. The command `ESC[`*r*`;`*c*`H` moves the cursor to row *r* and column *c*, so the command `ESC[1;1H` moves the cursor to the top left corner of the screen and `ESC[24,80H` moves it to the bottom right corner. One note: the lowercase f works identically to the uppercase H, so `ESC[10;10H` and `ESC[10;10f` do the same thing.

Ansi also has the ability to save the current cursor position with the command `ESC[s`. This command can only save one cursor position, so each time you use it, you overwrite the prior position it has saved. The `ESC[u` command returns to the position saved with the `ESC[s` command. You can also have Ansi report the cursor

position with an ESC[6n command. This is generally used by programs and is of little value in a batch file.

Video control

The Mode command allows you to control the video mode, but Ansi does an even better job. For example, the ESC[=1h command configures the screen for 40×25 color mode so that text appears very large. The ESC[=3h returns the screen to 80×25 mode. The format of the command is slightly unusual because it has an equal sign in the middle. Assuming your display supports them, the Ansi commands to change the display mode are as follows:

ESC[=0h	40×25 monochrome text.
ESC[=1h	40×25 color text.
ESC[=2h	80×25 monochrome text.
ESC[=3h	80×25 color text.
ESC[=4h	320×200 4-color graphics.
ESC[=5h	320×200 monochrome graphics.
ESC[=6h	640×200 2-color graphics.
ESC[=13h	320×200 16-color graphics. (EGA and VGA only.)
ESC[=14h	640×200 16-color graphics. (EGA and VGA only.)
ESC[=15h	640×350 monochrome graphics. (EGA and VGA only.)
ESC[=16h	640×350 16-color graphics. (EGA and VGA only.)
ESC[=17h	640×480 2-color graphics. (VGA only.)
ESC[=18h	640×480 16-color graphics. (VGA only.)
ESC[=19h	320×200 256-color graphics. (VGA only.)

Ansi also has several commands to control other aspects of how the screen appears:

ESC[=7h	Causes lines longer than 80 characters to wrap around to the next line. This is the default mode.
ESC[=7l	Causes lines longer than 80 characters to be truncated so they do not wrap. Note that the last character is a lowercase L and not a 1.
ESC[0m	Turns off all screen attributes, returning the screen to white on black.
ESC[1m	Turns on high-intensity foreground colors.
ESC[2m	Returns the foreground colors to normal intensity.
ESC[4m	Turns on underlining on a monochrome display.
ESC[5m	Causes the foreground text to blink.
ESC[7m	Converts the display to inverse video.
ESC[8m	Blanks the screen by setting the foreground color to the background color.

As you can see, Ansi offers a lot of control over video attributes.

Remember: Ansi offers a lot of control over video attributes. ☑

Keyboard control

Ansi is also able to reconfigure your keyboard. This topic is unrelated to batch files, so I will cover it only very briefly. The general format of the command is:

```
ESC[__;__p
```

The first blank is the ASCII value of the key to reassign. The second blank is the ASCII value the key is to take on. If you like, you can enclose the actual value in double quotes. For example, most of the time in DOS, you use the parentheses more

than the brackets (although the left bracket gets a workout under Ansi). You can switch them in DOS using the command:

```
ECHO ESC[91;40p
ECHO ESC[93;41p
ECHO ESC[40;91p
ECHO ESC[41;93p
```

The reassignment works under DOS but not in programs like Microsoft Word that bypass DOS for keystrokes. To reset the keyboard reassignments, issue the command:

```
ECHO ESC[91;91p
ECHO ESC[93;93p
ECHO ESC[40;40p
ECHO ESC[41;41p
```

You could have started to remap these using the actual keystrokes. So the first command would have been

```
ECHO ESC["[";"("p
```

However, after remapping the square brackets to parentheses, you would not have been able to type them into the second two commands to map the parentheses to square brackets because they have been turned into something else. Using ASCII values avoids this problem.

The disk that comes with this book has two batch files that will help you experiment with Ansi keyboard reassignments. The first is ANSIKEY1.BAT. It takes pairs of numbers associated with the various keystrokes: the first number is the keystroke to modify, and the second keystroke is its new value. It issues the Ansi command necessary to make the first assignment, issues two SHIFT commands, and checks to see if another pair of numbers is waiting. It's shown next without internal documentation and command line help:

```
:TOP
 IF (%2)==() GOTO END
 ECHO [%1;%2p
 SHIFT
 SHIFT
GOTO TOP
```

To remap the square brackets and parentheses mentioned earlier, you would enter the command

```
ANSIKEY1 91 40 93 41 40 91 41 93
```

and to restore the keys to their original value, you would enter

```
ANSIKEY1 40 40 41 41 91 91 93 93
```

The second batch file is ANSIKEY2.BAT. It is similar to ANSIKEY1.BAT, only it accepts the actual keystrokes rather than the numbers. It is shown next without internal documentation and command line help:

```
:TOP
 IF (%2)==() GOTO END
 ECHO ["%1";"%2"p
```

```
 SHIFT
 SHIFT
GOTO TOP
```

To remap the square brackets and parentheses mentioned earlier, you would enter this command:

```
ANSIKEY1 ( [ ) ] [ ( ] )
```

Because none of the keys are remapped until this command is issued, you have no trouble issuing all the commands on the command line. To restore the keys to their original value, you would enter

```
ANSIKEY1 ( ( ) ) [ [ ] ]
```

Both batch files will accept over twenty-five reassignments on a single line, making it easy to reassign a lot of keys at once. If you find yourself using the same reassignment command more than once or twice, you might want to either write a custom batch file to issue the Ansi commands or just to store the commands you use to call one of these batch files.

In addition to remapping standard keys, you can also remap extended keystrokes like Ctrl-F1. These keystrokes have a two-digit code that starts with "0;"—the number "zero," not the letter "O." They are entered like any other keystroke; thus, to remap Ctrl-F1 to issue a DIR command, you would issue the command:

```
ECHO ESC[0;94;"DIR";13p
```

Because this type of assignment involves one number and one character string, you cannot issue this command with either ANSIKEY1.BAT or ANSIKEY2.BAT, although either could easily be modified to handle this. TABLE 21-3 shows many of the ASCII and extended keystrokes you can use.

Table 21-3

Keystroke	ANSI Key Code	Keystroke	ANSI Key Code	Keystroke	ANSI Key Code
Alt-;	0;131	Alt-L	0;38	Del	0;83
Alt-0	0;129	Alt-M	0;50	Down Arrow	0;80
Alt-1	0;120	Alt-N	0;49	End	0;79
Alt-2	0;121	Alt-O	0;24	F1	0;59
Alt-3	0;122	Alt-P	0;25	F10	0;68
Alt-4	0;123	Alt-Q	0;16	F2	0;60
Alt-5	0;124	Alt-R	0;19	F3	0;61
Alt-6	0;125	Alt-S	0;31	F4	0;62
Alt-7	0;126	Alt-T	0;20	F5	0;63
Alt-8	0;127	Alt-U	0;22	F6	0;64
Alt-9	0;128	Alt-V	0;47	F7	0;65
Alt-A	0;30	Alt-W	0;17	F8	0;66
Alt-B	0;48	Alt-X	0;45	F9	0;67
Alt-C	0;46	Alt-Y	0;21	Home	0;71
Alt-D	0;32	Alt-Z	0;44	Ins	0;82

Table 21-3 Continued.

Keystroke	ANSI Key Code	Keystroke	ANSI Key Code	Keystroke	ANSI Key Code
Alt-dash	0;130	Ctrl-F1	0;94	Left Arrow	0;75
Alt-E	0;18	Ctrl-Left Arrow	0;115	NUL	0;3
Alt-F	0;33	Ctrl-Right Arrow	0;116	PgDn	0;81
Alt-F1	0;104	Ctrl-End	0;117	PgUp	0;73
Alt-F10	0;113	Ctrl-F10	0;103	Right Arrow	0;77
Alt-F2	0;105	Ctrl-F2	0;95	Shift-F1	0;84
Alt-F3	0;106	Ctrl-F3	0;96	Shift-F10	0;93
Alt-F4	0;107	Ctrl-F4	0;97	Shift-F2	0;85
Alt-F5	0;108	Ctrl-F5	0;98	Shift-F3	0;86
Alt-F6	0;109	Ctrl-F6	0;99	Shift-F4	0;87
Alt-F7	0;110	Ctrl-F7	0;100	Shift-F5	0;88
Alt-F8	0;111	Ctrl-F8	0;101	Shift-F6	0;89
Alt-F9	0;112	Ctrl-F9	0;102	Shift-F7	0;90
Alt-G	0;34	Ctrl-Home	0;119	Shift-F8	0;91
Alt-H	0;35	Ctrl-PgDn	0;118	Shift-F9	0;92
Alt-I	0;23	Ctrl-PgUp	0;132	Shift-Tab	0;15
Alt-J	0;36	Ctrl-Print Screen	0;114	Up Arrow	0;72
Alt-K	0;37				

As you can see, Ansi gives you a great deal of control over the screen. It is also difficult to learn; once you learn Ansi, though, it is fairly easy to use. TABLE 21-4 summarizes many frequently used Ansi commands.

An example

We can use Ansi keyboard remapping to construct a very useful batch file. Often, computer magazines print Debug scripts for you to enter to create small utilities. Because these are in hexadecimal, the numbers you must enter range 0–9 and A–F but not all of these are on your numeric keypad. ENTERHEX.BAT remaps the "/", "*", "–", "+", "=" and "Del" keys that surround the number pad as A–F so you can quickly enter the letters as well as the numbers. While this remapping works with Edlin and COPY CON, it does not work inside the DOS 5/6 EDIT program. The code for the operational portion of ENTERHEX.BAT follows:

```
ECHO Esc["/";"A"p
ECHO Esc["*";"B"p
ECHO Esc["-";"C"p
ECHO Esc["+";"D"p
ECHO Esc["=";"E"p
ECHO Esc[0;83;"F"p
ECHO Number Pad Ready For Hex Entry
```

UNENTER.BAT will restore the keys to normal. The code for the operational portion follows:

```
ECHO Esc["/";"/"p
ECHO Esc["*";"*"p
ECHO Esc["-";"-"p
ECHO Esc["+";"+"p
ECHO Esc["=";"="p
ECHO Esc[0;83;0;83p
ECHO Number Pad Back To Normal
```

Table 21-4

ANSI Command Sequence	Function
ESC[#;#p	Reassign the key defined by the first pound sign to the value of the second pound sign.
ESC[#A	Move the cursor up the number of rows specified by the number that replaces the pound sign.
ESC[#B	Move the cursor down the number of rows specified by the number that replaces the pound sign.
ESC[#C	Move the cursor left the number of rows specified by the number that replaces the pound sign.
ESC[#D	Move the cursor right the number of rows specified by the number that replaces the pound sign.
ESC[=0h	Set the display mode to 40x25 monochrome text.
ESC[=13h	Set the display mode to 320x200 16-color graphics.
ESC[=14h	Set the display mode to 640x200 16-color graphics.
ESC[=15h	Set the display mode to 640x350 monochrome graphics.
ESC[=16h	Set the display mode to 640x350 16-color graphics.
ESC[=17h	Set the display mode to 640x480 2-color graphics.
ESC[=18h	Set the display mode to 640x480 16-color graphics.
ESC[=19h	Set the display mode to 320x200 256-color graphics.
ESC[=1h	Set the display mode to 40x25 color text.
ESC[=2h	Set the display mode to 80x25 monochrome text.
ESC[=3h	Set the display mode to 80x25 color text.
ESC[=4h	Set the display mode to 300x200 4-color graphics.
ESC[=5h	Set the display mode to 320x200 monochrome graphics.
ESC[=6h	Set the display mode to 640x200 2-color graphics.
ESC[=7h	Turn line wrap on.
ESC[=7l	Turn line wrap off.
ESC[?7h	Turn line wrap on. This is needed only if line wrap has been turned off.
ESC[?7l	Turn line wrap off. When line wrap is off, once the cursor reaches the end of the line, each succeeding character is printed on top of the last one at the end of the line.
ESC[0h	Set the display mode to 40x25 monochrome.
ESC[0J	Erase the screen from the cursor position to the end of the screen. (Not all versions of Ansi support this command.)
ESC[0K	Erase the current line from the cursor position right.
ESC[0l	Set the display mode to 40x25 monochrome.
ESC[0m	Reset the screen display to white on black.
ESC[1h	Set the display mode to 40x25 color.
ESC[1J	Erase the screen from the cursor position to the top of the screen. (Not all versions of Ansi support this command.)

Table 21-4 Continued.

ANSI Command Sequence	Function
ESC[1K	Erase the line the cursor is on, from the beginning of the line to the cursor position. (Not all versions of Ansi support this.)
ESC[1l	Set the display mode to 40x25 color.
ESC[1m	Set the foreground text to bold.
ESC[2h	Set the display mode to 80x25 monochrome.
ESC[2J	Clear the screen.
ESC[2K	Erase the entire line that the cursor is on. (Not all versions of Ansi support this.)
ESC[2l	Set the display mode to 80x25 monochrome.
ESC[2m	Return the foreground colors to normal intensity.
ESC[30m	Set foreground color to black.
ESC[31m	Set foreground color to red.
ESC[32m	Set foreground color to green.
ESC[33m	Set foreground color to yellow.
ESC[34m	Set foreground color to blue.
ESC[35m	Set foreground color to magenta.
ESC[36m	Set foreground color to cyan.
ESC[37m	Set foreground color to white.
ESC[3h	Set the display mode to 80x25 color.
ESC[3l	Set the display mode to 80x25 color.
ESC[40m	Set background color to black.
ESC[41m	Set background color to red.
ESC[42m	Set background color to green.
ESC[43m	Set background color to yellow.
ESC[44m	Set background color to blue.
ESC[45m	Set background color to magenta.
ESC[46m	Set background color to cyan.
ESC[47m	Set background color to white.
ESC[4h	Set the display mode to 320x200 color graphics.
ESC[4l	Set the display mode to 320x200 color graphics.
ESC[4m	Set the foreground text to underlined on a monochrome display. On a color display, this sets the foreground to blue.
ESC[5A	Set the foreground text to blinking.
ESC[5h	Set the display mode to 320x200 monochrome graphics.
ESC[5l	Set the display mode to 320x200 monochrome graphics.
ESC[5m	Set the foreground color to blinking.
ESC[6h	Set the display mode to 640x200 monochrome graphics.
ESC[6l	Set the display mode to 640x200 monochrome graphics.
ESC[6n	Report the cursor position.
ESC[7m	Set the display to reverse video, black text on a white background.
ESC[K	Erase the current line from the cursor position right.
ESC[r;cf	Position the cursor on row number r and column number c.
ESC[r;cH	Position the cursor on row number r and column number c.
ESC[s	Store the current cursor position.
ESC[u	Return to the cursor position stored by the ESC[s command.
ESC8m	Blank the screen by setting the foreground color equal to the background color.

Summary

You can display a screen full of information by using Ansi commands.

Ansi is used to control the screen, cursor, video, and keyboard.

Ansi escape sequences can be used to change the foreground and background colors of the screen.

When Ansi is used to change the color of the screen, a CLS command does not reset the colors to white-on-black.

Ansi commands are called escape sequences.

Ansi commands cannot be entered from the keyboard because they usually start with the escape character.

If your word processor cannot enter an escape character in a file, pipe the CLS command to a file and incorporate the escape from that file in your document.

Ansi processes any line that is sent to the screen that begins with an escape character followed by an open bracket, no matter how the characters are displayed.

Ansi escape sequences can be displayed using the ECHO command.

Ansi escape sequences can be displayed by storing them in an ASCII file and typing that file to the screen.

Ansi escape sequences can be displayed by storing them in environmental variables and then using the SET command or ECHO %variable% command to display one or more environmental variables.

Ansi escape sequences can be displayed using the PROMPT command, which has a built-in escape character.

When Ansi escape sequences are displayed in a batch file via the PROMPT command, command echoing must be turned on while the prompt is being used to send the Ansi commands.

Ansi provides several different ways to erase different portions of the screen.

Ansi provides the ability to move the cursor in any direction as well as to save and recall the cursor position.

Ansi can be used to set the video mode.

Ansi also allows you to remap the keyboard, although the remapping will not work with many programs that avoid DOS to get their keystrokes.

22 Batch file subroutines

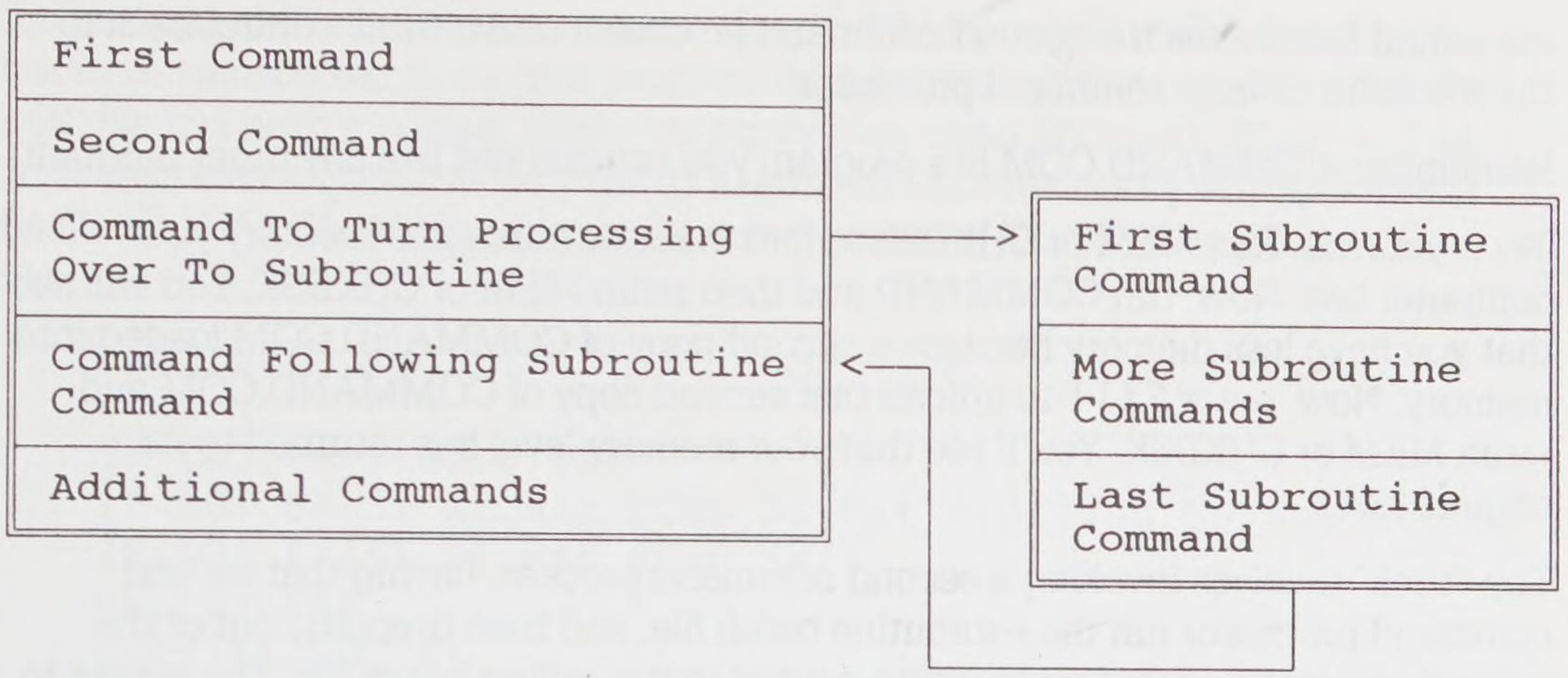
In programming, a subroutine is a special type of program. When a program calls a subroutine, control is passed to the subroutine. When the subroutine is finished, control is passed back to the program that called the subroutine. The calling program continues from the point where it passed control to the subroutine. Figure 22-1 illustrates this graphically.

Subroutines are a way of writing a section of code once, storing it in a special way, and reusing it as many times as it is needed. In a batch file, this often makes the logic much easier to follow and the batch file easier to write and debug. For example, if you want to perform multiple commands based on the errorlevel value, you must have a series of IF-tests for each possible errorlevel value. Each IF-test must jump to a different section of the batch file to perform the multiple commands, and then the sections must reconverge. With subroutines, the batch file would have a single command for each IF-test where that command ran a subroutine. Look at this example:

```
IF ERRORLEVEL 3 SUB-3
IF ERRORLEVEL 2 SUB-2
IF ERRORLEVEL 1 SUB-1
IF ERRORLEVEL 0 SUB-0
```

Of course, as we learn to write subroutines, we will replace the commands after the errorlevel-tests with the actual commands to run subroutines.

Remember: Subroutines allow you to reuse your debugged batch file code.



22-1
When one program or part of a program calls another program, the part being called is named a subroutine.

What's a subroutine, and how do I write one?

A subroutine is nothing more than one or more lines of batch file commands—exactly like any other batch file commands. The question is not "What is it?" but rather "How do you use it?" Your options are as follows:

- You can store the subroutine as a separate batch file. All versions of DOS support this, although the methods of running these separate batch files differ slightly depending on the version.
- You can store the subroutine inside the original batch files and use IF-tests to jump to this section of code.

As a separate batch file

The easiest way to write a subroutine is to write that subroutine as a stand-alone batch file. The method's biggest advantage is that it allows multiple batch files to call on a common subroutine; its worst disadvantage is that it uses more hard disk space because each batch file is allocated a full cluster of space no matter how small it is. Of course, this is not an issue if you use disk compression.

Remember: The easiest way to write a subroutine is to write that subroutine as a stand-alone batch file.

Prior to DOS 3.3

Early versions of DOS (prior to 3.3) did not support a batch file running other batch files as subroutines. With those older versions of the operating system, once a batch file passed control to another batch file, control never passed back to the original batch file.

This is not how a batch file behaves when it passes control to a program. When that program finishes, control is passed back to the calling batch file. This fact can be used to "trick" earlier versions of DOS into allowing one batch file to call another and gain back control when the subroutine batch file finishes.

While it is possible to "trick" older versions of DOS into treating other batch files as subroutines—and I'll show you how to do that in a moment—my suggestion is to simply upgrade to DOS 6. This new version has enough new and improved features that the upgrade will be well worth your time and money.

Normally, you do not think of COMMAND.COM as a program; yet, it is. Running COMMAND.COM by entering COMMAND at the command line prompt runs COMMAND.COM. This loads a second copy of the command processor into memory. The command to exit this second command processor is EXIT; this

command terminates the second command processor and returns control back to the first (and calling) command processor.

Remember: COMMAND.COM is a program you can run just like any other program.

Try it yourself. Run MEM or CHKDSK to find out how much free memory your computer has. Now, run COMMAND and then rerun MEM or CHKDSK. You will see that you have less memory because a second copy of COMMAND.COM loaded into memory. Now, enter EXIT to unload that second copy of COMMAND.COM and rerun MEM or CHKDSK. You'll see that your memory level has returned to its original level.

The "trick" involves invoking a second command process, having that second command processor run the subroutine batch file, and then dropping out of the second command processor to return control to the calling batch file. The syntax to run the second batch file is

```
COMMAND /C Batch
```

The /C tells COMMAND.COM to run the following command, which in this case is a batch file. The last command of the subroutine batch file should be an EXIT. This returns control to the original batch file.

DOS 3.3 and later

DOS 3.3 added the CALL batch subcommand. Thus, a batch file can call another batch file with

```
CALL Batch
```

Once the subroutine batch file terminates, control automatically returns to the batch file that called the subroutine batch file. The subroutine batch file does not have to end with an EXIT command. Users with DOS 3.3 or later should always use the CALL command and never use the COMMAND/C trick since using CALL allows the batch files to share a common environment and minimizes memory requirements.

Remember: The CALL command makes it easy for one batch file to run another batch file and regain control when it terminates.

An example

To illustrate this, we will look back to TOA.BAT. TOA.BAT was a batch file we began looking at in Chapter 10 as a way to copy .DOC and .CMP files to the A drive and delete the associated .BAK file when the user entered just the filename. This batch file was expanded over the next several chapters, reaching its ultimate form in Chapter 16. That version of TOA.BAT is shown here:

```
@ECHO OFF
:TOP
 IF (%1)==() GOTO END
 DEL %1.BAK
 COPY %1.DOC A:
 COPY %1.CMP A:
 SHIFT
 GOTO TOP
:END
```

The same batch file but with internal documentation and command line help is included on the disk that comes with this book.

The purpose of this batch file is to copy the modified versions of files to a floppy disk to be transported to another location. As it exists, it does not enforce this. If a .DOC file has been modified, then—on my system using Microsoft Word—a .BAK file always exists. Suppose we wanted to add the logic necessary to skip copying the file if a .BAK file does not exist. Our new version of TOA.BAT might look like this:

```
@ECHO OFF
:TOP
 IF (%1)==() GOTO END
 IF NOT EXIST %1.BAK ECHO %1 Not Modified So Not Copied
 IF NOT EXIST %1.BAK GOTO SKIP
 DEL %1.BAK
 COPY %1.DOC A:
 COPY %1.CMP A:
:SKIP
 SHIFT
 GOTO TOP
:END
```

Now, suppose we wanted to add logic to the copy portion of the batch file to detect minor copying errors—like a full diskette—and warn the user. Plus, we wanted to make sure the user entered the name right by warning him or her if the .DOC file does not exist. As you can imagine, the original three lines of the batch file that handled the repetitive tasks of copying and deleting files, shown next, are going to become very complex.

```
DEL %1.BAK
COPY %1.DOC A:
COPY %1.CMP A:
```

Predictably, because this chapter covers subroutines, the solution is to place these three commands in a separate batch file and treat them as a subroutine. The original version of TOA.BAT (called TOA-2.BAT on the disk) has these three lines stripped out and a single command added to call the subroutine. Within TOA-2.BAT, the files to be copied are known as %1. However, this replaceable parameter is only global while inside TOA-2.BAT and is not available to the subroutine—called TOA-2SUB.BAT. In order to make the name of the file available to TOA-2SUB.BAT, we must pass it to that batch file as a replaceable parameter. The resulting TOA-2.BAT looks like this:

```
@ECHO OFF
:TOP
 IF (%1)==() GOTO END
 CALL TOA-2SUB %1
 SHIFT
 GOTO TOP
:END
```

A version of TOA-2.BAT with internal documentation and command line help is included on the disk that comes with this book.

As we begin looking at TOA-2SUB.BAT, it looks like this:

```
@ECHO OFF
DEL %1.BAK
COPY %1.DOC A:
COPY %1.CMP A:
```

Two notes about this subroutine are important. First, it does not need to test to see if %1 exists because TOA-2.BAT takes care of that prior to calling the subroutine. Second, the @ECHO OFF command is not really necessary because command echoing is always off when a batch file with command echoing turned off calls another batch file as a subroutine. You will see why I've added it later.

Because this subroutine is shorter and not already convoluted by the looping performed in TOA.BAT, we can add the additional checks we desired without the batch file becoming as complicated as TOA.BAT would have become. First, let's add the logic to test and make sure the file has been modified by testing to see if the .BAK file exists. The pseudocode for this is

If .BAK does not exist, tell user and exist subroutine
If .BAK exists, continue normally

The resulting subroutine looks like this:

```
@ECHO OFF
IF NOT EXIST %1.BAK ECHO File Not Modified—Not Copied
IF NOT EXIST %1.BAK GOTO END
DEL %1.BAK
COPY %1.DOC A:
COPY %1.CMP A:
:END
```

Next, we want to make sure the file actually exists before we try to copy it. If the file was created by Lotus or dBASE, a .BAK file might still exist but a .DOC file might not. If the .DOC file does not exist, we want to exit without copying anything; but if the .CMP file does not exist, we simply want to skip that COPY command because not all documents have .CMP files. The pseudocode for this is

If .DOC file does not exist, warn user and exit subroutine
If the .CMP file exists, copy it to the diskette

and the resulting subroutine looks like this:

```
@ECHO OFF
IF NOT EXIST %1.BAK ECHO File Not Modified—Not Copied
IF NOT EXIST %1.BAK GOTO END
IF NOT EXIST %1.DOC ECHO %1.DOC File Not Found
IF NOT EXIST %1.DOC GOTO END
DEL %1.BAK
COPY %1.DOC A:
IF EXIST %1.CMP COPY %1.CMP A:
:END
```

Because the second set of IF-tests makes sure the .DOC file exists, no IF-test is used in front of that COPY command. However, when it is time to copy the .CMP file, the batch file has not verified that it exists, so this COPY command is preceded by an IF-test.

As our last modification to the logic, we want to make sure that the file was successfully copied to the A drive. Because COPY does not set the errorlevel, we must perform this test manually. The pseudocode for this is

If file is on current drive but not A drive warn user

The resulting subroutine looks like this:

```
@ECHO OFF
IF NOT EXIST %1.BAK ECHO File Not Modified—Not Copied
IF NOT EXIST %1.BAK GOTO END
IF NOT EXIST %1.DOC ECHO %1.DOC File Not Found
IF NOT EXIST %1.DOC GOTO END
DEL %1.BAK
COPY %1.DOC A:
IF EXIST %1.CMP COPY %1.CMP A:
IF NOT EXIST A:%1.DOC ECHO %1.DOC Not Copied
Successfully
:END
```

This test will spot occasions when the file was too big to fit on the A drive; however, it won't handle situations where there was a version of the file already on the A drive and some other problem prevented the successful copying, all because the batch file will find that A:%1.DOC exists. While COPY does not set an errorlevel value, XCOPY does. A 0 represents a successful copy, while a higher value represents some sort of problem. We can use that information to modify this subroutine to spot any copy problem by using XCOPY and errorlevel tests. The resulting subroutine looks like this:

```
@ECHO OFF
IF NOT EXIST %1.BAK ECHO File Not Modified—Not Copied
IF NOT EXIST %1.BAK GOTO END
IF NOT EXIST %1.DOC ECHO %1.DOC File Not Found
IF NOT EXIST %1.DOC GOTO END
DEL %1.BAK
IF EXIST %1.CMP XCOPY %1.CMP A:
XCOPY %1.DOC A:
IF ERRORLEVEL 1 ECHO %1.DOC Not Copied Successfully
:END
```

In addition to converting the COPY commands to XCOPY and converting the final IF-EXIST test to IF-ERRORLEVEL, I also reversed the copying order of the .CMP and .DOC files. The errorlevel used in the IF-ERRORLEVEL test is the final errorlevel value, and the .DOC document file is far more important than the .CMP dictionary file. So, by copying the .DOC file last, its errorlevel is the one tested. Interested readers might want to add tests on both values if the first XCOPY has a problem.

These are all the modifications the subroutine needs if it is to function solely as a stand-alone subroutine. However, because it exists as a stand-alone batch file, users are likely to try and run it to see what it does. We can help avoid problems with this by adding command line help when the user uses the now familiar /? switch, but this time also display the help if the batch file is started without a replaceable parameter. This final test works because TOA-2.BAT always calls it with a replaceable parameter; the batch file knows it is being run improperly if it does not receive a replaceable parameter. The final version of the batch file is shown next:

```
@ECHO OFF
IF (%1)==()   GOTO HELP
IF (%1)==(/?) GOTO HELP
IF NOT EXIST %1.BAK ECHO File Has Not Been Modified--Not
Copied
IF NOT EXIST %1.BAK GOTO END
IF NOT EXIST %1.DOC ECHO %1.DOC File Not Found
IF NOT EXIST %1.DOC GOTO END
DEL %1.BAK
IF EXIST %1.CMP XCOPY %1.CMP A:
XCOPY %1.DOC A:
IF ERRORLEVEL 1 ECHO %1.DOC Not Copied Successfully
GOTO END
:HELP
 ECHO This Batch File Is A Subroutine Used By TOA-2.BAT
 ECHO It Is Not Designed To Be Run By The User
GOTO END
:END
```

This version of TOA-2SUB.BAT is included on the disk that comes with the book. The only difference is that the disk version includes some additional internal documentation.

By now, you should also see why I included the `@ECHO OFF` command in the subroutine, although it's not needed when the batch file is running as a subroutine. If a user runs the batch file from the command line to find out what it does, the help section will take care of that, but everything would look very unattractive without the `@ECHO OFF` command.

Inside the calling batch file

Each batch file you write takes up a minimum of one cluster of disk space. Depending on several factors, that can be up to 8K and will most likely be at least 2K—unless you are using disk compression. If you are running very short of hard disk space, that can seem like a large price to pay for a relatively small subroutine. You can avoid this by bundling all the subroutines within the main batch file.

When you include the subroutine(s) inside the batch file, it ends up running itself. A program or batch file that runs itself is said to be *recursive*. Recursive batch files have two advantages over batch files that call separate batch files as subroutines. First, as mentioned earlier, the resulting batch file is generally smaller than the separate batch file and subroutines. This is generally true in spite of the

requirement in the recursive batch file for additional IF-tests to direct it to the appropriate spot in the batch file to continue processing.

Remember: A program or batch file that runs itself is said to be recursive.

The second advantage of a recursive batch file is how easy it is to transfer it to different systems. If you need to copy a recursive batch file to another system, then that batch file is all you need to take to the new machine. If you want to transfer a batch file with external subroutines, then you have to remember to copy all the files—something I have trouble remembering to do.

There is, however, one very significant drawback to recursive batch files. The resulting batch file is longer and more complex and, without the proper approach, can be much more difficult to write. This additional complexity is an important consideration—one you should not overlook.

Warning: Building the subroutine inside the main batch file saves space but makes the resulting batch file more complex.

Storing subroutine commands within a batch file

A replaceable parameter functions as a signal to tell the batch file if it is being called by itself as a subroutine. When that signal is missing, the batch file knows it is being called from the command line. A series of IF-tests at the beginning of the batch file direct it to the appropriate section of the batch file depending on the flag it receives as a replaceable parameter. That way, the batch file can contain more than one subroutine internally.

We'll start with TOA-2.BAT as written earlier, and we'll include TOA-2SUB.BAT as an internal subroutine. We'll call the resulting batch file TOA-3.BAT. TOA-3.BAT needs to run exactly like TOA-2.BAT when it first starts. However, when TOA-2.BAT would be calling TOA-2SUB.BAT as a subroutine, TOA-3.BAT needs to call itself. We will use %2 as a flag to TOA-3.BAT that it is running as a subroutine—specifically, when TOA-3.BAT calls itself as a subroutine, it will pass "XYZ123ABC" as %2.

That means we have to have several IF-tests at the top of TOA-3.BAT. One will direct the batch file to help if /? is entered, and another will direct the batch file to the subroutine section if %2 is XYZ123ABC. Because TOA-3.BAT will be providing this replaceable parameter, capitalization will not be a problem. The end of the subroutine section will be a `GOTO END` command, so the subroutine copy of the batch file will exit and return control to the calling copy of TOA-3.BAT. Otherwise, the subroutine section is identical. The resulting batch file is shown next:

```
@ECHO OFF
:TOP
 IF (%1)==()   GOTO END
 IF (%1)==(/?) GOTO HELP
 IF (%2)==(XYZ123ABC) GOTO SUB
 CALL TOA-3 %1 XYZ123ABC
 SHIFT
GOTO TOP
:HELP
 ECHO This Batch File Requires You To Enter Filenames
 ECHO Without The Extension. For Each Name, It Erases
 ECHO The Associated .BAT File Then It Copies The
```

```
 ECHO Associated .DOC And .CMP Files To The A Drive
GOTO END
:SUB
 IF NOT EXIST %1.BAK ECHO File Not Modified—Not Copied
 IF NOT EXIST %1.BAK GOTO END
 IF NOT EXIST %1.DOC ECHO %1.DOC File Not Found
 IF NOT EXIST %1.DOC GOTO END
 DEL %1.BAK
 IF EXIST %1.CMP XCOPY %1.CMP A:
 XCOPY %1.DOC A:
 IF ERRORLEVEL 1 ECHO %1.DOC Not Copied Successfully
GOTO END
:END
```

TOA-3.BAT calls itself as a subroutine. It also illustrates the best approach to writing recursive batch files. Because the individual-files approach is easier to write, that's what I used here. Only after TOA-2.BAT and TOA-2SUB.BAT were finished did I begin working on the single recursive batch file that incorporated both files.

This example also illustrates the reason for a habit you might be wondering about, one that I've developed when writing batch files. I have always closed out the last section of the batch file—generally the help section—with a GOTO END command, even when the next line was the :END label. By doing that, I avoid introducing problems when I rearrange the logic of the batch file—as I did when I created TOA-3.BAT by modifying TOA-2.BAT to include the subroutine.

It is also possible to reorder the IF-tests in TOA-3.BAT to make it run more quickly. Because TOA-3.BAT calls itself as a subroutine for each file, it is likely to run as a subroutine more often than as a stand-alone batch file. As written, when running as a subroutine, it has to first pass through the two IF-tests relating to command line help. By putting the subroutine IF-test first, these two IF-tests will be avoided when TOA-3.BAT runs as a subroutine. In general, when using branching IF-tests in a batch file, they should be ordered from the most likely to be true to the least likely.

Remember: When using multiple IF-tests to branch in a batch file, arrange them so the ones most likely to be true are the first in the series.

Summary

A subroutine is reusable code that one or more batch files can access when needed to perform some specific function.

Subroutines can be stored as separate batch files and run with the CALL command.

Storing subroutines as stand-alone batch files is the easiest way to write and debug subroutines; but it takes up more disk space, and you must remember to provide all the subroutines when you give a batch file to another user.

When using the CALL command to access another batch file, that called batch file does not have access to the replaceable parameters of the original batch file, so it must be passed any information it needs.

It's important that stand-alone batch file subroutines protect themselves from being run by users from the command line. The safest form of protection is to test for a

specific replaceable parameter provided by the batch file that is supposed to call the subroutine batch file.

Storing the subroutine inside the original batch file saves space and makes it easier to transport the batch file, but it also makes the resulting batch file more complex.

Subroutines can be stored inside the original batch file in two ways while remaining compatible with MS DOS and batch file compilers: by using a series of If-tests to direct the batch file to the appropriate location each time it starts, and by using a replaceable parameter as a flag when calling the batch file as a subroutine.

If you are writing a subroutine that can possibly be called by more than one batch file, then it makes a lot of sense to create that subroutine as a stand-alone batch file. That way, all the batch files can access that single subroutine. If the subroutine will only be used by one batch file, then it makes more sense to create that subroutine as part of the original batch file.

23 When batch files don't work

By now, you should have a good understanding of the basics of writing and using batch files. You should be able to create a batch file that uses standard DOS commands as well as batch-specific commands and even some Batcmd commands. You should also know how to use replaceable parameters and If-tests to control the flow of the batch file. As you begin to use this knowledge in your computing, however, you're going to occasionally have a batch file that doesn't work properly—so you need to know what to do.

If your batch file is in a loop or doesn't appear to be doing anything, hit ^C or ^Break. This should stop the batch file and give you a "Terminate batch job (Y/N)?" prompt. Answer Y for yes and press Enter. This is the only way to stop a batch file. If it doesn't work, you'll have to reboot. Once you get it stopped, you need to locate the problem. You can list the file on screen with the command:

```
TYPE FILE.BAT
```

However, you'll need to use an editor to make changes, so you might as well load it into an editor to begin with.

While it is not possible to give you a checklist of everything you should do when your batch file fails, what follows is a list of what I've found to be the most common troubleshooting approaches I've found useful.

Possible error messages

If you are lucky, the operating system will tell you why your batch file is not working properly. However, the message is likely to be cryptic and difficult to understand. Listed next are some of the error messages you are likely to see, along with their causes and possible solutions. The wording sometimes changes slightly

between different versions and brands of DOS, so be sure to check for messages similar to the ones you are getting and not just for an exact match.

Syntax Error

This is the nastiest error to deal with because it gives you so little information. Generally, a command like an IF-test is using the wrong syntax. The solution is to turn command echoing on so that you can see which line is the problem line and then review the syntax for that command to see where the problem is. Far and away, the two most common problems here are using a single equal sign for the IF-test and testing a replaceable parameter with a test like

```
IF %1==A
```

without first making sure the replaceable parameter has a value. You can avoid this error by always surrounding the replaceable parameter you are testing and the test value with parentheses, so there is never a nul value on either side of the test. Thus, this test should be rewritten as

```
IF (%1)==(A)
```

Bad Command or Filename

Command or Filename Not Recognized

The batch file is trying to run a program that is not in the current subdirectory or path. Usually, this is caused by misspelling the name of a program or batch subcommand or by forgetting to change subdirectories or drives before running a program from a subdirectory not in your path. The solution is to verify the spelling of the program or batch subcommand or make sure the batch file changes to the proper drive and subdirectory. You should always have a batch file change to the proper drive before changing to a subdirectory because you never know for sure if the user has changed drives before running the batch file.

Label Not Found

The batch file issued a GOTO command, but the label listed after GOTO does not exist. The solution is to rewrite the batch file to include the specified label.

Abort, Retry, Ignore, Fail?
Abort, Ignore, Retry
Abort, Retry, Fail
Data Error
Drive Not Ready
Physical Media Error
Sector Not Found

The batch file or a program being run by the batch file is unable to read from or write to the drive it is trying to use. The most common cause is trying to read from or write to a floppy disk drive without a diskette in it, without the drive door shut, or with an unformatted diskette in the drive. The second most common cause is a defective disk or diskette.

If you forgot to insert a diskette, go ahead and insert it and then press Retry. However, you should never change disks while one is being read from or written to. For any other problem, press Fail and troubleshoot from the operating system. If you have a defective diskette, replace it. If you have a defective hard disk, you can generally correct the problem with a utility program like the Norton Utilities or PC Tools.

Access Denied

The file is a read-only file. If the file is read-only and you want to modify it, you must change the read-only flag with the ATTRIB command.

Batch File Missing
Batch file . . . missing retry [Y/N]?

DOS only reads batch files one line at a time. After executing that line, it goes back to the batch file to read the next line. When DOS cannot find the batch file to read the next line, you get this error message. The most common cause of this error message is a batch file that deletes itself. For example, BATCH.BAT

```
@ECHO OFF
DEL BATCH.BAT
DIR
```

will generate this error message because when DOS reads the DEL BATCH.BAT line, it knows there is another line; but after deleting the batch file, it goes back to read the next line and the batch file is not there. Other less common reasons for this error message are batch files that rename themselves, substituting or unsubstituting drives containing the batch file, or swapping a floppy disk that contains the batch file for a different disk.

The solution is simple. Review the logic of the batch file and make sure that nothing in the batch file causes the operating system to "lose" the batch file. It's usually easy to find the problem line. Just turn command echoing on and look to see which is the last line the batch file executes; that's generally the problem line.

If you want a batch file to delete itself, you can have it do so without generating this error message; however, because the resulting batch file only runs once, it's usually easier to just ignore the error message. In order for the operating system to delete the batch file without complaining about it being missing, absolutely nothing must follow the DEL BATCH.BAT command—not even a return. The easiest way to make sure this is the case is to enter the batch file from the command line via the `COPY CON BATCH.BAT` command and press F6 or ^Z immediately after entering `DEL BATCH.BAT` without pressing Return first.

Batch Files Nested Too Deep

One batch file can call another with CALL. When that happens, the second batch file is said to be nested inside the first. That second batch file can then call a third, and so on. If these batch files become nested too deeply, they abort with the "Batch Files Nested Too Deep" error message. The solution is to reduce the number of nested batch files you have.

Not Enough Memory
Not Enough Memory For . . .
Not Enough Memory To . . .

The batch file is trying to load a program but doesn't have enough memory to run that program. The solution is to increase available memory by removing unnecessary device drivers or memory resident software.

Not Ready Error

The device the batch file is trying to access (e.g., LPT1, CON, or COM1) is not ready, the disk drive is not ready to use, or the hard disk has failed. For non-drive devices, try rebooting. If the problem continues, verify that the requested device is actually installed. If it is, the computer needs to be repaired. For a disk drive, try inserting a new and formatted disk and making sure the drive door is shut. For a hard disk, try using a utility program like PC Tools or the Norton Utilities to repair the drive. If you do not have experience in this area, get help.

Warning: Repairing a defective disk can be difficult. If you are not sure what you are doing, get experienced help.

Cannot Start Command, Exiting
No Free File Handles/Cannot Start Command, Exiting
Too Many Files Open
Too Many Open Files

The operating system is trying to open more files than is allowed. This generally happens when running a second batch file with the command

```
COMMAND/C Batch
```

rather than using CALL. The solution is to either rewrite the batch file or increase the values for the FILE switch in the CONFIG.SYS file and reboot.

Environment Full

The batch file is trying to create an environmental variable but doesn't have enough room. The solution is to either delete unwanted environmental variables or expand the size of the environment.

Environment Error

DOS cannot find the current environment. The only solution is to reboot.

Filename Too Long

DOS only supports filenames of eight characters or less and extensions of three characters or less. The solution is to shorten the filename to an acceptable length.

Disk Full

The batch file is trying to COPY or otherwise write to a disk that either has no more room or (when writing/copying to the root directory) has no more room in the file allocation table [FAT] for directory entries. The solution is to erase unwanted files.

Write Protect Error

The diskette the batch file or application is trying to write to is write-protected. The solution is to remove the write-protection.

File Sharing Conflict

This error only occurs on a network. The action of the batch file or application running from within the batch file conflicts with actions of another network user. For example, trying to delete a file that is in use by another user. The solution is to rewrite the batch file to avoid conflicting with other users.

On Or Off Parameter Required

The batch file issued a BREAK following by something other than an On/Off switch. These two switches, or using BREAK by itself, are the only three valid ways of issuing a BREAK command. The solution is to rewrite the batch file to use BREAK properly.

Invalid Path Or Path Not Found
Invalid Path Specified: Check Path

The batch file has issued a command that required DOS to search the path. While searching the path, DOS found an invalid entry in the path. The solution is to make sure your path contains only valid subdirectories.

Other

Many other error messages are possible because any application your batch file runs can itself trigger an error message from the operating system. In addition, the applications the batch file is running might themselves display an error message. When you see an error message you do not recognize while your batch file is running, check the back of your DOS manual to see if it is listed there along with possible solutions. If not, check the manual for the software your batch file is currently running to see if it lists that error message and possible solutions.

Check the spelling

The next thing to check is the spelling. It's very easy to make a spelling error, especially when you're copying the file from the console. Sometimes it is visually difficult to spot a word that is slightly misspelled, like ECHE for ECHO. I use Microsoft Word, a processor that lets you specify your own dictionaries, so I've developed a dictionary of batch commands and the common DOS commands and messages I use in batch files. When I think a typo or spelling error might exist, I run the batch file through the spelling checker in Word using this dictionary. If your dictionary allows custom dictionaries, you might try the same thing.

If your word processor does not allow custom dictionaries or you do not want to go through the trouble of creating one, you might try an old proofreading trick. When reading anything forwards, one tends to get interested in the content and anticipate the words or spelling that should be there. That makes it harder to catch errors. The solution is to read backwards—or in the case of batch files, where lines usually cannot be split up, from the last line to the first. While you will not be able to follow the logic (that's the idea), you will be much more likely to spot syntax errors.

Look closer

Sometimes, it's hard to follow the execution of a batch file. With command echoing turned off and a fast computer, it can be very hard to follow your batch files as they run. When I have this problem, I remove the @ECHO OFF line and slow my computer down to its slowest speed before running the batch file. If the commands

still go by too fast, I load Windows and run the batch file from inside Windows. Usually, that slows down the batch file enough so I can watch it.

Isolate the problem

As you begin taking a closer look at your batch file, you will most likely find that most of it runs fine but breaks down at some particular point. Once I've isolated the problem to a particular section of the batch file, I turn off command echoing for the rest of the batch file and just turn it on for that section by using an ECHO ON command at the beginning of the section and an @ECHO OFF at the end. I also insert several PAUSE commands in the problem section so that I can take my time examining the information on the screen while the problem section runs.

Split out the problem section

Occasionally, a particular section just seems so wrong that nothing I do seems to make it work right. When that happens, I remove the problem section and make it a separate batch file. That way, I can run it by itself—supplying the inputs it needs—without the rest of the batch file getting in the way. I can add ECHO commands to see the values of variables and any number of PAUSE commands to review the screen. When I finally get this batch file working, I merge it back into the original batch file.

Diagram the logic

When you have a problem involving logic testing, the best solution to resolving the problem is to draw a "map" of the logic of the batch file. That way, you can make sure all the paths end up where they should and none of them end up "dangling" without reaching a proper conclusion. It's been my experience that these dangling paths are one of the major sources of logic errors. Figures 23-1 and 23-2 show a map of the logic of COPYTHE2.BAT, the solution to Problem 9 in Chapter 24.

Put it aside for a while

Sometimes, when nothing seems to work, the best thing you can do is put the batch file aside for a while and come back to it. A number of times, I've put aside a batch file with what seemed to be an impossible task only to come back to it later and see an obvious problem. This approach works for many problems, not just batch files.

Start over

If all else fails, the final solution is to erase the problem batch file or batch file segment and start over. Often, this second attempt will turn out better than the first because you have experience with the problem already.

Summary

If your batch file generates an error message, that message can help you find the problem with your batch file.

If your batch file appears to lock up, you can often regain control by pressing Ctrl-Break.

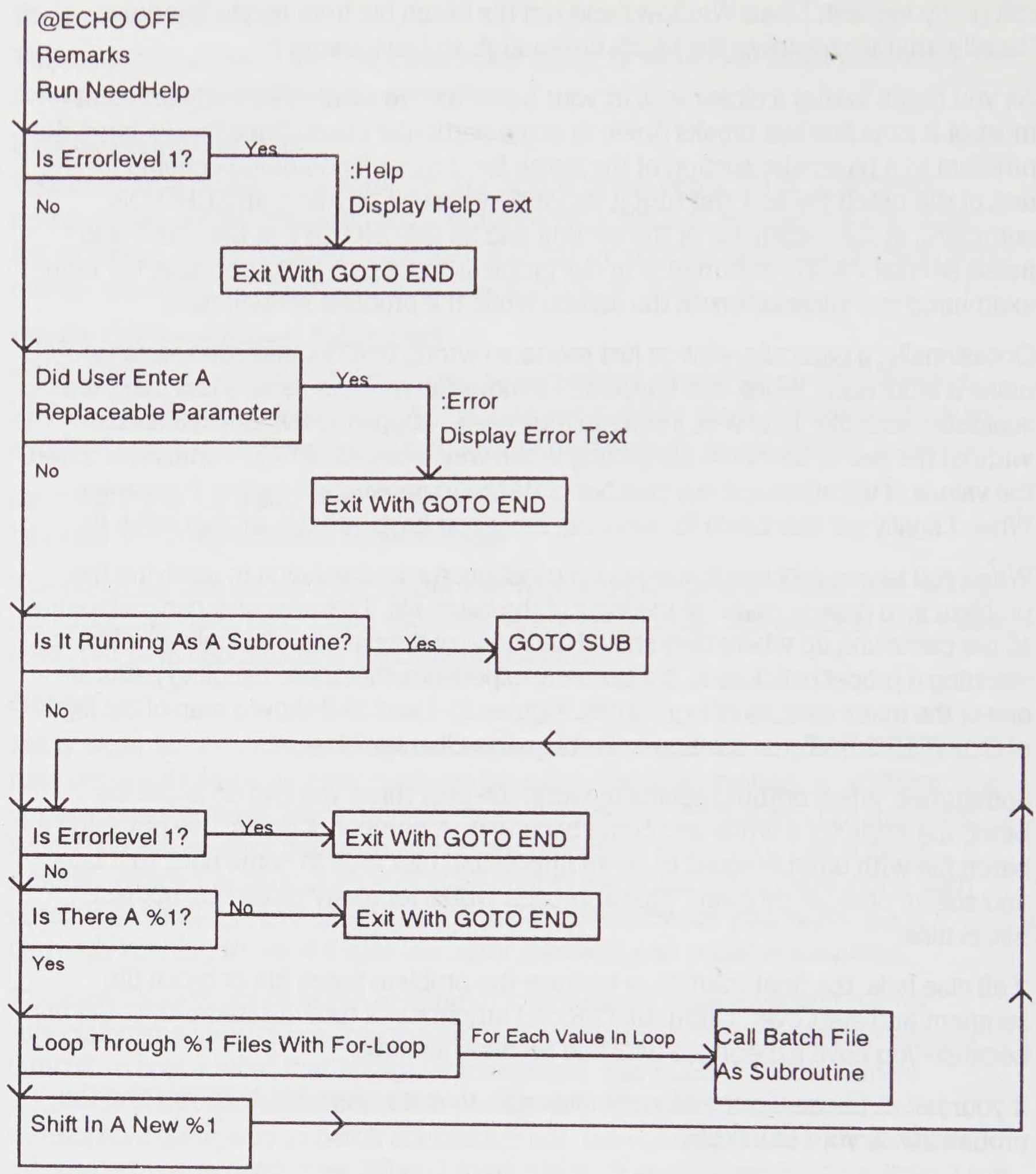
After looking for error messages, the next thing to check when a batch file does not work is to check the spelling of the commands.

When checking the spelling does not resolve the problem, try turning command echoing on and adding PAUSE commands to watch command execution closer.

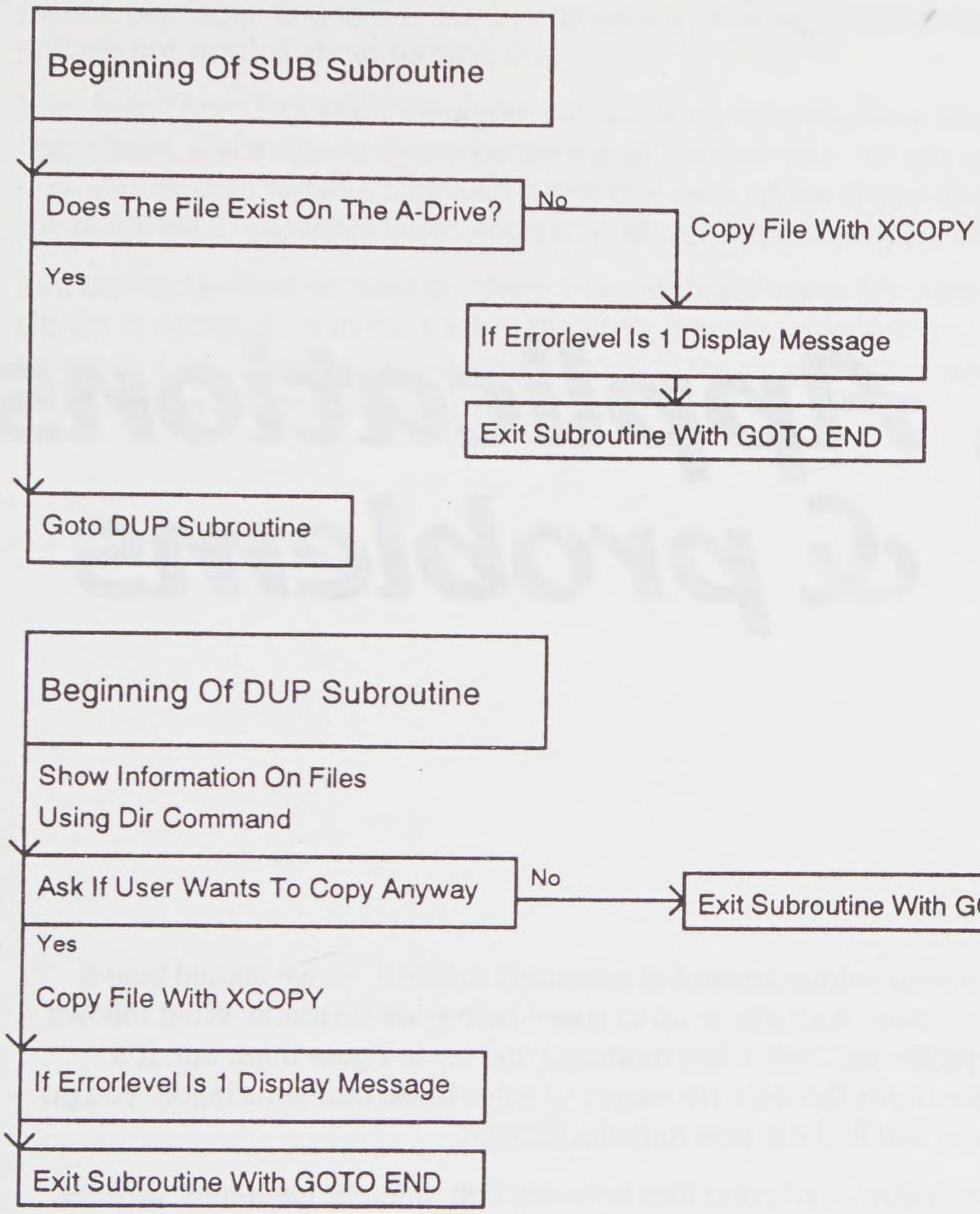
When a closer review does not resolve the problem, try isolating the problem portion of the batch file into a separate batch file.

When the problem with a batch file involves logic testing, try drawing a map of the logic.

As a last resort, try putting the problem batch file aside for a while or even starting over.



23-1 *A diagram of the logic of the main portion of COPYTHE2.BAT.*



23-2 *A diagram of the logic of the subroutine portion of COPYTHE2.BAT.*

Applications & problems

We've covered a tremendous amount of advanced material, so we should take a few moments and make sure you're up to speed before we continue. What follows are three quick problems. Take a few moments and try to figure them out. It's important that you have the skills necessary to solve these problems before you go on; otherwise, you will find the new material difficult.

Problem 7

A friend of yours is always copying files between two different machines. Several times, she has made the mistake of copying a file to a floppy disk when another version of that very file was already on the floppy disk. She has asked you to write a batch file to copy all the files she specifies on the command line to the A drive—but only if the file does not already exist on the A drive.

The complete file name with extension will be provided on the command line, so no assumptions about the name should be made. When wildcards are used, the batch file should copy all the files to the A drive that do not exist there while skipping the ones already on the A drive. Also, keep the user informed of the process.

Problem 8

You are working with a program that you do not normally use, and you find yourself entering the same long command over and over. You will only be using this program for a few weeks, so you first elected not to construct a batch file to issue this one command. However, you've run into this circumstance before, so you change your mind and decide to write a batch file to handle it.

When you run this new batch file with a command after its name, the batch file will store that command in the environment and then run it. When you run the batch file alone, it will issue the command stored in the environment. Some of your commands are very long, so you want to avoid limiting the commands to nine

pieces, corresponding to %1–%9. You do have a lot of free environmental space so you are not worried about running out.

Problem 9

Your friend from Problem 7 likes your batch file but asks you for a modification. Sometimes, she wants to overwrite the file on the disk with the one on the hard disk. Rather than just skipping all the files that exist on the floppy disk, she wants the batch file to show her information on both versions and ask her what to do.

Answers

You can tackle each of these problems in a number of ways. My solutions are shown in Appendix A in the back of the book. It is not important that you match my answers; if you developed a working solution to the problem, your solution is just as good as mine and maybe even better.

Part five

Improving batch files with software

As useful as batch files are, software can make them even more useful. We have already looked at several handy utilities. This section looks at three major software packages that can make your batch files even more useful.

Batcmd: New batch file commands In order to write really powerful batch files, you need a batch file utility to provide some commands that DOS does not offer. This book includes a very powerful batch utility called Batch Commands, or Batcmd for short. You have already seen some of its power from Chapter 19. This chapter documents the full power of Batcmd.

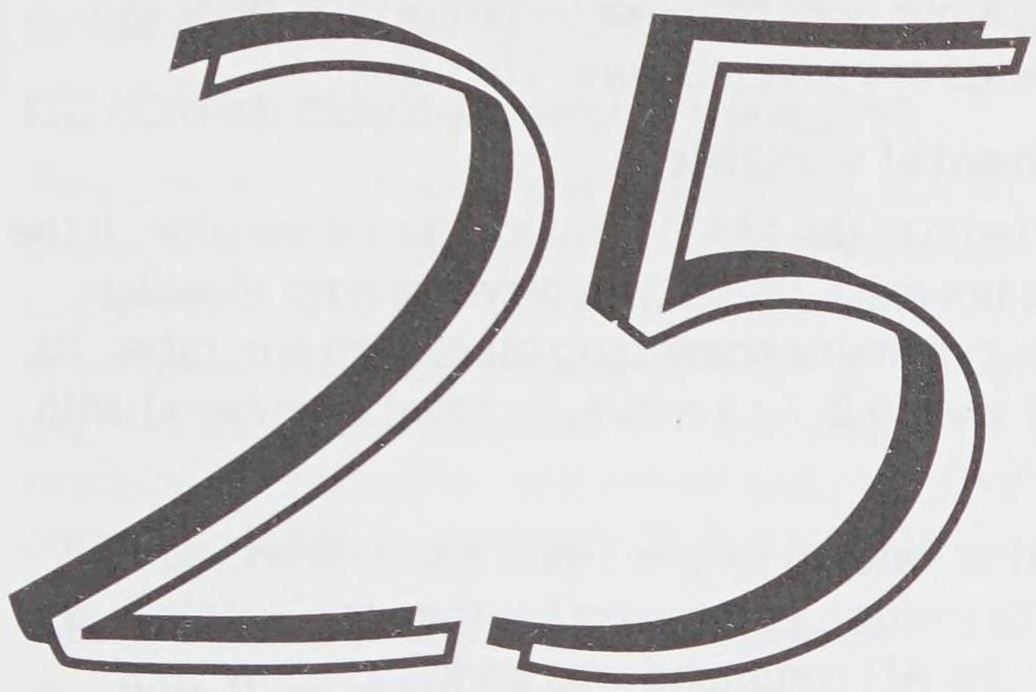
Chapter 25

Other utilities In addition to Batcmd, this book includes several other utility programs. This chapter covers these remaining utilities.

Chapter 26

Builder and Builder Lite When you decide that batch files are not powerful enough for your needs, the next step up is a batch file compiler. One of the most powerful and easy-to-use batch file compilers available is Builder from hyperkinetix. If you want to try a less powerful version, hyperkinetix, TAB Books, and myself have combined to bring you Builder Lite.

Chapter 27



Batcmd
New batch file commands

When I started writing about batch files, I was very frustrated by their inability to perform some of the simplest tasks, like asking the user a question. However, many utilities are now available to get around the limitations of batch files—so many, in fact, that I was able to write a whole book describing many of them.

That still left me with the problem of writing about batch files that needed one or more of these utilities. I could not very well dictate which ones for the reader to buy; and without knowing what utilities the reader already had, writing a batch file any reader could use was impossible. So I sat down and designed a batch file utility that would perform those functions I needed in my batch books.

Once I had designed the program, I turned that design over to Doug Amaral of hyperkinetix. (hyperkinetix is the company that wrote Builder and Builder Lite, two batch file compilers.) Doug kindly wrote Batcmd (short for "Batch Commands") especially for my batch file books. Thanks, Doug!

Batcmd is free with the book, so I expect that most of you will end up using it. However, keep in mind that, with all of these batch files, you can substitute any other batch utility that performs the same function. You do not have to use Batcmd.

The general format of the command is

```
BATCMD Keyword <Prompt>
```

where *Keyword* is a two-letter command abbreviation that tells Batcmd what to do and *Prompt* is an optional message to the user what information to enter. The `Prompt` can contain multiple words and should not be enclosed in quotation

marks. A few of the commands require additional information as explained later under the specific command discussion.

Most of the Batcmd keywords cause Batcmd to either get information from the user or from the computer. In most cases, that information is passed back to the batch file through the errorlevel.

The Batcmd commands for getting input from the user were covered back in Chapter 19. The remaining commands are covered here. They are:

AD (ADd 1 to MATH environmental variable)

This command adds one to the contents of the MATH environmental variable. If the MATH environmental variable does not exist, Batcmd displays an error message and does nothing. If MATH exists but contains something other than a number, its contents are treated as though they were a 0. As a result, its value is replaced with a 1 (0 + 1).

The AD command was not included in Batcmd to give batch files mathematical power. In fact, it would be difficult to perform anything other than the simplest math via the AD command. Rather, the AD command was added to allow your batch files to go through loops a specific number of times. For example, you might ask the user for a password with the GE and use the AD command to make sure the user only gets three attempts at the password.

BE (BEep the speaker)

This command beeps the speaker. If the command is entered as BATCMD BE, then it beeps the speaker once. If it is entered as BATCMD BE n, then it will beep the speaker n times. Once started, this beeping cannot be aborted without rebooting so Batcmd limits n to 99 or less. If you have a specific need to beep the speaker more than 99 times, you can repeatedly run `BATCMD BE n` or include it in a FOR-loop.

BS (Blank the Screen)

This command clears the screen, identically to the DOS CLS command. However, if you enter colors after the command, it clears the screen to those colors. The command format is `BATCMD BS Foreground Background`. TABLE 25-1 shows the color numbers that can be used with the BS command. All sixteen numbers can be used for the foreground color, but only the first eight can be used for the background color.

Table 25-1

Number	Color	Number	Color
0	Black	8	Gray
1	Blue	9	Bright Blue
2	Green	10	Bright Green
3	Cyan	11	Bright Cyan
4	Red	12	Bright Red
5	Magenta	13	Bright Magenta
6	Brown	14	Yellow
7	White	15	Bright White

CC (Check Conventional memory)

This command checks to see how much conventional memory is available and returns through errorlevel the number of 32K blocks; an errorlevel value of 10 would indicate that 320K is free. You would use this command to make sure enough memory was available if a program being run by a batch file was particularly sensitive to available memory. Keep in mind that the available memory is reduced by the memory used by Batcmd before this measurement is made.

CE (Check Environmental memory)

This command checks to see how much environmental space is free and returns through errorlevel the number of free bytes. Because errorlevel is limited to an integer from 0 to 255, this command returns 255 for any environment where the free environmental space exceeds 255 bytes. This is a good way for batch files that depend on the environment to check before running to make sure space is available. In particular, you might want to use the CE command before the GE command if the information GE is storing in the environment is critical.

CH (Check number of Hard drives)

This command checks to see how many hard drives are installed. It returns a 0 if there are no hard drives; otherwise, it sets errorlevel equal to the number of hard drives. This command checks the number of physical hard drives, not the number of partitions. Thus, a 100M drive that is partitioned into three 32M partitions and a fourth 4M partition will still count as one physical drive to Batcmd. It does not include any CD ROM drives or other non-standard drives in the calculation.

CL (Check LIM memory)

This command checks to see how much LIM memory is free and returns with an errorlevel equal to the number of 32K blocks of expanded memory. This is useful as a check prior to running a program that requires a minimum amount of LIM memory.

CM (Check Mouse)

This command sets errorlevel to 1 if it finds a mouse and to a 0 otherwise. Some older programs need a special command line switch in order to use a mouse; this command lets the batch file check for the presence of a mouse when it runs rather than forcing the batch file to behave in a certain fashion.

CS (Check Space)

This command checks to see how much free space is available on a drive and returns the number of 32K blocks of free space in errorlevel. When free space exceeds 8,160K, it returns 255. If the command is entered as `BATCMD CS`, it uses the default drive. If the command is entered as `BATCMD CS` *`drive`*, then it checks the amount of free space on the specified drive. This command is especially useful before running a database program where you will be working with large files because many database programs create a lot of working files on the disk.

EX (EXit)

This command is entered as `BATCMD EX` *n*, where *n* is a number from 0 to 255. Batcmd sets the errorlevel to the number specified and exits. If no number is

specified, Batcmd sets errorlevel to 0. If a fractional number is specified, Batcmd uses only the integer portion of the number, so BATCMD EX 33.3 would result in an errorlevel value of 33. If a number over 255 is specified, 256 is repeatedly subtracted from the specified number until the resulting number falls between 0 and 255, the only valid values for errorlevel. If a negative number is specified, 256 is repeatedly added to the specified number until the resulting number is between 0 and 255.

One use for this command is to give your batch files the ability to report their results via the errorlevel, just as programs do. Another use is to reset the errorlevel to 0. Sometimes when working with subroutines, it is useful to put errorlevel-testing at the top of a loop—as COPYTHIS.BAT (the solution to Problem 7 in Chapter 24) does. However, this can cause problems if a program run before this batch file has set errorlevel because the first time through the loop, COPYTHIS.BAT has not yet run XCOPY, so errorlevel still contains the value from the last program. Adding a `BATCMD EX 0` to the top of any batch file using errorlevel testing avoids this problem.

PC (Position Cursor)

This command positions the cursor at the specified coordinates on the screen. The format of the command is `BATCMD PC row column`, where *row* is the row number to move the cursor to (1–24) and *column* is the column number to move the cursor to (1–80). If a non-integer row or column number is used, the fractional portion of the number is ignored. If a 0, negative number, or number outside the acceptable boundaries is used, Batcmd aborts with an error message and sets the errorlevel to 1.

This command replicates the ability of ANSI to position the cursor without needing ANSI. Being able to position the cursor can make it easier to draw complex screens and can help keep the screen uncluttered.

RB (Reboot the Computer)

This command performs a warm reboot of the computer, which can be useful if you are writing a batch file to change the configuration of the computer or as a way of aborting a password batch file after a specific number of attempts.

SD (Store Day)

This command stores the day of the month in errorlevel. If the date were March 21, `BATCMD SD` would set errorlevel to 21. This command—and the date and time commands that follow—are useful for performing specific tasks on a specific day. For example, you might perform an incremental backup every Friday from the AUTOEXEC.BAT and a full backup the first of every month.

SH (Store Hour)

This command stores the hour of the day, in 24-hour format, in errorlevel. If the time is 6:30PM, `BATCMD SH` sets errorlevel to 18.

SL (Skip Line)

This command causes a batch file to skip a single line on the screen, just like echoing an Alt-255.

SM (Store Minute)

This command stores the minute of the hour in errorlevel. If the time is 6:30PM, `BATCMD SM` sets errorlevel to 30.

SO (Store mOnth)

This command stores the month of the year in errorlevel. If the date is March 21, `BATCMD SO` would set errorlevel to 3.

SS (Store Second)

This command stores the second in errorlevel. Of course, this information changes rapidly, but errorlevel is only updated each time the command is run. If the time is 6:30:15, `BATCMD SS` sets errorlevel to 15.

SU (SUbtract 1 from MATH environmental variable)

This command subtracts one from the contents of the Math environmental variable. If the Math environmental variable does not exist, Batcmd displays an error message and does nothing. If Math exists but contains something other than a number, its contents are treated as though they were a zero. Like the AD command explained earlier, SU is designed for looping and not really for adding mathematical power to batch files.

SY (Store Year)

This command stores the last two digits of the year in errorlevel, so 1992 is stored as 92 and 2000 is stored as 0.

VO (VOlume label)

This command stores the volume label of the specified drive in the environment under the BATCMD environmental variable.

Summary

TABLE 25-2 shows a brief summary of the Batcmd keywords and their function.

Table 25-2

Command abbreviation	Command name	Function
AD	ADd	Add one to the number stored in the MATH environmental variable.
BE	BEep	Beep the speaker.
BS	BlankScreen	Clear the screen, optionally using the colors specified after the BS command.
CC	CheckConventional	Return the amount of free conventional memory in 32K blocks.
CE	CheckEnvironment	Return with an errorlevel equal to the amount of free environmental space up to 255 bytes, setting it to 255 for that amount and over.
CH	CheckHard	Return with an errorlevel 0 if no hard disk is installed, otherwise set the errorlevel equal to the number of logical hard disk drives installed.
CL	CheckLim	Return with an errorlevel equal to the number of 32K blocks of expanded memory.

Table 25-2 Continued

Command abbreviation	Command name	Function
CL	CheckLim	Return with an errorlevel equal to the number of 32K blocks of expanded memory.
CM	CheckMouse	Return with an errorlevel of 1 if there is a mouse, 0 otherwise.
CS	CheckSpace	Return with an errorlevel equal to the number of 32K blocks free on the specified drive. If no drive specified, use the default.
EX	EXit	Set the errorlevel to the number specified after the EX and exit.
GE	GetEnvironment	Display an optional prompt and accept a multi-character input from the user and store it in the environment under the name BATCMD .
GF	GetFromlist	Display an optional prompt, accept an input only if it is contained in the list following the prompt, exit with an errorlevel equal to one for the first item in list, two for the second, and so on.
GK	GetKey	Display an optional prompt, accept any keystroke, and exit with the ASCII value of that keystroke.
GL	GetLetter	Display an optional prompt, accept a letter, and exit with an errorlevel of 1 for A, 2 for B, and so on. If the user presses any other keystroke, beep and continue waiting for a number.
GN	GetNumber	Display an optional prompt, accept a single-digit number, and exit with an errorlevel equal to that number. If the user presses any other keystroke, beep and continue waiting for a number.
GU	GetUpper	Display an optional prompt, accept an uppercase letter, and exit with an errorlevel equal to the ASCII value of the letter. If the user presses a lowercase letter or any non-letter keystroke, beep and continue waiting for an uppercase letter.
PC	PositionCursor	Position the cursor on the row and column specified after the PC.
RB	ReBoot	Reboot the computer.
SD	StoreDay	Return with an errorlevel equal to day of month.
SH	ShowHour	Return with an errorlevel equal to the hour of the day, in 24-hour format.
SL	SkipLine	Skip one blank line on the screen.
SM	StoreMinute	Return with an errorlevel equal to minute of hour.

Table 25-2 Continued

Command abbreviation	Command name	Function
SO	StoremOnth	Return with an errorlevel equal to month.
SS	StoreSecond	Return with an errorlevel equal to the second.
SU	SUbtract	Subtract one from the number stored in the MATH environmental variable.
SY	StoreYear	Return with an errorlevel equal to the last two digits of the year. If possible, have it work properly for 2000 and beyond.
VO	VOlume Label	Store the volume label of the specified drive in the environment.

SHOWGETS.BAT (look back at FIG. 19-1) illustrates using Batcmd to obtain information from the user and then acting on that information.

DOBATCMD.BAT (see FIG. 25-1) shows using most of the remaining commands in a batch file to obtain information from the computer.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: DOBATCMD.BAT`
`REM PURPOSE: Illustrate Non-Input`
`REM Batcmd Commands`
`REM VERSION: 1.00`
`REM DATE: May 27, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`SET SPACE=`	The environmental variable Space contains ten spaces and is used several times to erase text from a particular line.
`BATCMD BS 15 1`	Run Batcmd and use the BS (Blank Screen) option to clear the screen to bright white on blue.
`ECHO Creating Math Environmental Variable`	Tell the user what is happening.
`SET MATH=0`	Create the Math Environmental variable.
`ECHO Will Now Count To Fifty`	Tell the user what is happening.
`PAUSE`	Pause the program until the user presses a key.

25-1
DOBATCMD.BAT shows how to use most of the Batcmd commands not involved in obtaining information from the user.

Batch File Line	Explanation
BATCMD BS 13 1	Use the Batcmd BS option to clear the screen to bright magenta on blue.
:TOPAD	Label marking the top of a loop.
BATCMD PC 12 40	Use the Batcmd PC (Position Cursor) command to position the cursor on the screen.
ECHO %MATH%	Display the current value of the Math environmental variable. Since "Batcmd PC" is used before this command each loop, it always appears at the same location.
BATCMD AD	Use the Batcmd AD (Add) command to increment the Math environmental variable by one.
IF NOT %MATH%==51 GOTO TOPAD	Until Math reaches 51, continue looping.
BATCMD PC 22 01	Position the cursor.
ECHO Now Will Count Down To Zero	Tell the user what is happening.
SET MATH=50	Reset Math to 50, since the above loop went past 50 to 51.
PAUSE	Pause the program until the user presses a key.
BATCMD BS 12 1	Clear the screen to bright red on blue.
:TOPSU	Label marking the top of a loop.
BATCMD PC 12 40	Position the cursor.
IF %MATH%==9 ECHO IF %MATH%==9 BATCMD PC 12 40	It has a problem going from 10 to 9 since the nine is written over the one but the zero stays on the screen, resulting in a 90. These lines correct the problem by echoing two blank spaces (Alt-255) to erase the "10".
ECHO %MATH%	Display the current value of the Math variable.
BATCMD SU	Decrease the Math variable by one using The Batcmd SU (Subtract) command.
IF NOT %MATH%==0 GOTO TOPSU	If the value of Math has not yet reached zero, continue looping.

25-1
Continued.

Batch File Line	Explanation
PAUSE	Pause the program until the user presses a key.
SET MATH=	Reset this environmental variable since it is no longer needed.
BATCMD BS 15 1	Clear the screen to bright white on blue.
BATCMD PC 05 01	Position the cursor.
ECHO Beeping Speeker Five Times	Tell the user what is happening.
BATCMD BE 5	Beep the speaker five times using the Batcmd BE (Beep) command.
BATCMD PC 07 01	Position the cursor.
BATCMD CC	Check available memory using the Batcmd CC (Check Conventional) command.
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment under the name Error. Note that SAY-ERROR.BAT must be in the path for DOBATCMD.BAT to run properly.
BATCMD PC 07 01	SAYERROR.BAT displays a single line of information on the line the cursor was positioned on with the Batcmd PC command. By repositioning the cursor on the same line, the ECHO command that follows overwrites the SAYERROR-.BAT message.
ECHO %ERROR% 32K Blocks of Conventional Memory	Display the results of the Batcmd CC command.
BATCMD PC 09 01	Position the cursor.
BATCMD CE	Check the amount of environmental memory using the Batcmd CE (Check Environmental) command.
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment.
BATCMD PC 09 01	Position the cursor.
ECHO %ERROR% Bytes Of Environmental Memory	Display the results of the Batcmd CE command.
BATCMD PC 11 01	Position the cursor.

25-1
Continued.

Batch File Line	Explanation
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment.
BATCMD PC 09 01	Position the cursor.
ECHO %ERROR% Bytes Of Environmental Memory	Display the results of the Batcmd CE command.
BATCMD PC 11 01	Position the cursor.
BATCMD CH	Check the number of hard disks using the Batcmd CH (Check number of Hard disk) commands.
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment.
BATCMD PC 11 01	Position the cursor.
ECHO %ERROR% Hard Disk(s)%SPACE%	Display the results of the Batcmd CH command.
BATCMD PC 13 01	Position the cursor.
BATCMD CL	Check the amount of LIM memory using the Batcmd CL (Check LIM memory) command.
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment.
BATCMD PC 13 01	Position the cursor.
ECHO %ERROR% 32K Blocks of LIM Memory	Display the results of the Batcmd CL command.
BATCMD PC 15 01	Position the cursor.
BATCMD CM	Check to see if a mouse is installed using the Batcmd CM (Check Mouse) command.
IF ERRORLEVEL 1 ECHO Mouse Found IF NOT ERRORLEVEL 1 ECHO No Mouse Found	Display the results of the Batcmd CM command.
BATCMD PC 17 01	Position the cursor.
BATCMD CS	Check the amount of free space on the current drive using the Batcmd CS (Check Space) command.
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment.
BATCMD PC 17 01	Position the cursor.
ECHO %ERROR% 32K Blocks of Free Space On Default Drive	Display the results of the Batcmd CS command.
BATCMD PC 19 01	Position the cursor.
PAUSE	Pause the program until the user presses a key.

25-1
Continued.

Batch File Line	Explanation
`BATCMD BS 15 1`	Clear the screen to bright white on blue.
`BATCMD PC 03 01`	Position the cursor.
`BATCMD SD`	Find the day of the month using the Batcmd SD (Store Day) command.
`CALL SAYERROR`	Run SAYERROR.BAT to store the errorlevel in the environment.
`BATCMD PC 03 01`	Position the cursor.
`ECHO Day Of Month Is %ERROR%`	Display the results of the Batcmd SD command.
`SET DAY=%ERROR%`	Store this information to another environmental variable to keep the value.
`BATCMD PC 05 01`	Position the cursor.
`BATCMD SO`	Store the month to the errorlevel using the Batcmd SO (Store mOnth) command.
`CALL SAYERROR`	Run SAYERROR.BAT to store the errorlevel in the environment.
`BATCMD PC 05 01`	Position the cursor.
`SET MONTH=%ERROR%`	Store this information to another environmental variable to keep the value.

`IF %MONTH%==1 ECHO Month Is January`
`IF %MONTH%==2 ECHO Month Is February`
`IF %MONTH%==3 ECHO Month Is March`
`IF %MONTH%==4 ECHO Month Is April`
`IF %MONTH%==5 ECHO Month Is May`
`IF %MONTH%==6 ECHO Month Is June`
`IF %MONTH%==7 ECHO Month Is July`
`IF %MONTH%==8 ECHO Month Is August`
`IF %MONTH%==9 ECHO Month Is September`
`IF %MONTH%==10 ECHO Month Is October`
`IF %MONTH%==11 ECHO Month Is November`
`IF %MONTH%==12 ECHO Month Is December`
	Display the results of the Batcmd SO command.

`BATCMD PC 07 01`	Position the cursor.
`BATCMD SY`	Store the year to the errorlevel with the Batcmd SY (Store Year) command.
`CALL SAYERROR`	Run SAYERROR.BAT to store the errorlevel in the environment.
`BATCMD PC 07 01`	Position the cursor.
`SET YEAR=19%ERROR%`	Store this information to another environmental variable to keep the value.

25-1
Continued.

25-1
Continued.

Batch File Line	Explanation
BATCMD SY	Store the year to the errorlevel with the Batcmd SY (Store Year) command.
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment.
BATCMD PC 07 01	Position the cursor.
SET YEAR=19%ERROR%	Store this information to another environmental variable to keep the value.
ECHO Year Is %YEAR%%SPACE%	Display the results of the Batcmd SY command. Since this message is shorter than the message from SAYERROR.BAT, the message displays the Space variable (ten spaces) at the end to erase the rest of the SAYERROR.BAT message.
BATCMD PC 09 01	Position the cursor.
ECHO Date Is %MONTH%/%DAY%/%YEAR%	Display the entire date using the environmental variables used to store this information.
SET MONTH= SET DAY= SET YEAR=	Delete these variables that are no longer needed.
BATCMD PC 11 01	Position the cursor.
BATCMD SH	Store the hour to errorlevel using the Batcmd SH (Store Hour) command.
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment.
BATCMD PC 11 01	Position the cursor.
ECHO Hour Is %ERROR%%SPACE%	Display the results of the Batcmd SH command.
SET HOUR=%ERROR%	Store this information to another environmental variable to keep the value.
BATCMD PC 13 01	Position the cursor.
BATCMD SM	Store the minute to errorlevel using the Batcmd SM (Store Minute) command.
CALL SAYERROR	Run SAYERROR.BAT to store the errorlevel in the environment.
BATCMD PC 13 01	Position the cursor.
ECHO Minute Is %ERROR%%SPACE%	Display the results of the Batcmd SM command.

Batch File Line	Explanation
`SET MINUTE=%ERROR%`	Store this information to another environmental variable to keep the value.
`BATCMD PC 15 01`	Position the cursor.
`BATCMD SS`	Store the second to the errorlevel using the Batcmd SS (Store Second) command.
`CALL SAYERROR`	Run SAYERROR.BAT to store the errorlevel in the environment.
`BATCMD PC 15 01`	Position the cursor.
`ECHO Second Is %ERROR%%SPACE%`	Display the results of the Batcmd SS command.
`SET SECOND=%ERROR%`	Store this information to another environmental variable to keep the value.
`BATCMD PC 17 01`	Position the cursor.
`ECHO Time Is %HOUR%:%MINUTE%:` `%SECOND%%SPACE%`	Display the entire time using the environmental variables used to store this information.

`SET HOUR=`
`SET MINUTE=`
`SET SECOND=`
`SET ERROR=`
	Delete these variables that are no longer needed.

`BATCMD SL` `BATCMD SL`	Skip two lines using the Batcmd SL (Skip Line) command.
`ECHO Leaving With Errorlevel Set To 240` `ECHO Run SAYERROR.BAT To Check`	Tell the user what is happening.
`BATCMD EX 240`	Set the errorlevel to 240 using the Batcmd EX (EXit) command.
`GOTO END`	Jump to the end and exit.

`:HELP`
`  ECHO Illustrate Non-Input`
`  ECHO Batcmd Commands`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

25-1
Continued.

26 Other utilities

The major utility that comes with this book is Batcmd and it was covered in the last chapter. However, several smaller utilities are also included. Those smaller utilities are documented in this chapter.

ADD This takes one number passed to it via the command line, adds one to it and returns the result. There are versions on the disk written in QBasic (\BAT\QB-ADD), Builder Lite (\BAT\BLD-ADD) and Pascal (\BAT\P-ADD). These are demonstration programs to illustrate how to write batch utilities in a particular language and their operation varies depending on the language used to write them. See Chapter 34 for more details.

ASCII.EXE This program was designed to aid you in incorporating high-ordered ASCII characters in your batch file. When you enter "ASCII" from the command line, it displays all the printable ASCII characters in one screen so you can easily find the character you are looking for. Its memory requirements are minimal so you can generally run ASCII.EXE while shelled out of another program.

BS BS is short for BatScreen. BS lets you take an ASCII text file and turn it into a colored .COM program file that displays the text file on the screen very quickly. Full details on BS are given in Chapter 20.

CAPITAL CAPITAL takes a single word as input, converts that word to all uppercase and stores the resulting word in the environment under the environmental variable named "RONNY." This makes it easy to test replaceable parameters without worrying about capitalization, just run them through CAPITAL first and the resulting environmental variable is always uppercase. CAPITAL only accepts one word as an input and sets errorlevel to one if it encounters a problem.

ISITWIN

Some actions, like CHKDSK/F, are ok under DOS but unacceptable under Windows—even if you are shelled out and running DOS. ISITWIN gives your batch files a way of knowing if they are running Windows. When run, if Windows is running, it sets errorlevel to 1 even if shelled out to DOS. If Windows is not running, it sets errorlevel to 0.

ISITZERO

Oftentimes, it's useful to know if a file is a 0-length file. Usually, you pipe some information through the FIND filter and then pipe it to a file. Because FIND does not support errorlevel, your only way to find out if FIND found the information you asked for is to see if the resulting file contains any information. If FIND was unsuccessful, the file will be a 0-length file and MS-DOS will not copy 0-length files. Unfortunately, DR DOS copies them, so it's not possible to write batch files that use this technique and which will run under both operating systems. Plus, it's time-consuming to have the batch file copy a file to another name, test to see if the copy exists, and then delete both copies. ISITZERO overcomes this. The command `ISITZERO` *`file`* sets the errorlevel to 4 if *`file`* is a 0-length file and to 3 if *`file`* contains data. Because ISITZERO works under both DR and MS-DOS, it makes writing a common batch file for both systems easy plus it's quicker and neater than copying files.

MULTIPLY

This takes two numbers passed to it via the environment, multiplies them together, and returns the result. There are versions on the disk written in QBasic (\BAT\QB-TIMES), Builder Lite (\BAT\BLD-TIME), and Pascal (\BAT\P-TIMES). These are demonstration programs to illustrate how to write batch utilities in a particular language and their operation varies depending on the language used to write them. See Chapter 34 for more details.

NEEDHELP

One of the more difficult tests to perform in a batch file is to see if the user requested help with the "/?" switch because, unlike DR DOS, MS-DOS does not separate the batch file name and switch if they are entered without a space between them. NEEDHELP solves this problem. When run with the command `NEEDHELP %0 %1` it sets errorlevel to 1 if help was requested and to 0 otherwise.

27 Builder & Builder Lite

As powerful and useful as batch files are, when I work with them, I always have this nagging feeling that I could write really powerful batch files if the language could just do a little bit more. With a batch file, even simple activities require a utility program.

There are literally hundreds of utilities designed to enhance batch files, including the CHOICE utility included with DOS 6. Most of these programs, however, introduce a new headache if you're writing a batch file for use on another system: How do you ensure that the other system has the batch file utility you plan to use? If the utility is in the public domain, you can include a copy. If it's shareware, you can still include a copy, along with a note suggesting the user register the copy. However, if you need a commercial utility you're just plain stuck.

Even if you're able to send the user a copy of the utility, you still have the problem of managing the files so the user gets all the necessary files and puts the utility program(s) in a subdirectory in the path so that the batch file can use it.

So when you're writing batch files for other people, you must worry about which brand and version of operating system they have, plus you must worry about which batch utilities they have and what the syntax is. As a result, batch files intended for multiple users tend to end up being a real pain!

Notice: Writing batch files intended for multiple users on different platforms can end up being a real pain.

Multi-purpose batch languages

Builder

Several do-it-all batch languages, like Enhanced Batch Language and Batchman, can add an incredible amount of power to your batch files. But you still come back to the problem of trying to share those powerful batch files with other users. Unless those other users have the same utility, your fancy batch file will not run on their system.

What you need is a way of legally bundling the utilities you use together with the resulting batch file so you can distribute everything and not worry about what batch utilities the user has. Builder does just that. But Builder also does much more: it gives you the power and flexibility of a programming language, the speed of a compiled language, and full compatibility with the MS-DOS batch subcommands, as well as the ease of learning with this batch language.

Remember: With Builder, you can legally bundle the utilities you use together with the resulting batch file and distribute everything together as one program.

☑

Bundling

Builder converts a batch file into a stand-alone program through a process called *compiling*. Compiling means that all the commands are converted to instructions that the computer can understand—including commands that use all of the extensions Builder builds into the batch language. And because the resulting program does not need the Builder language to run—it's nothing but instructions that the computer already understands—you can distribute this program royalty-free, and everyone running your program can use all the language extensions you used when you wrote the program.

Because you're sending them everything they need as part of the program, you don't need to worry about what utilities they have. Because all the code is in one .EXE program, file management is easy. Plus, because you are distributing a program, you have the security of knowing that the user cannot change it.

Remember: Builder programs are powerful, easy to distribute and royalty free!

☑

Power & flexibility of a programming language

The main reason for using Builder is its many enhancements. To give you an idea of the power of Builder, the Batcmd utility that comes with this book was written using only a small subset of the features of Builder, and Batcmd is a very powerful extension to the batch language!

The NEEDHELP.EXE, ASCII.EXE, and CAPITAL.EXE programs that come with this book were also written in Builder. I wrote these three programs, and neither one took longer that thirty minutes to write, debug, and test! Anything you might want a batch file to do can be easily programmed into Builder. Additionally, Builder is so powerful that it generally has a command or function to do exactly what you want to do.

One of the most common uses of batch files is to write menus. Imagine if you were not limited to the simple menus batch files offer but could write fancy menus like the one used in the README.EXE program that comes on the disk included with this book. Take a moment to run this program. Notice that the menu gives you three ways to select an item from the menu:

- ❑ Move the cursor to that item and press Return.
- ❑ Press the highlighted letter for the item you want to run.
- ❑ Click with the mouse on the item you want to run.

Now notice that the menu has a shadow background, and notice the colors. The menu is in one color, the menu items are in another, and the screen background is still another color. This entire program was written and debugged in under an hour, using Builder!

Builder gives you far too many extensions to the batch language for me to list them all here, but these are some of the ones I like best:

- **Menus.** The menus in README.EXE are just the beginning. Builder can produce moving lightbar, popup, and dropdown menus, as well as a special menu that allows you to place individual menu items anywhere on the screen. You could use this to write a menu where option one is in the top left corner, option two is in the bottom right corner, and option three is in the middle.
- **User input.** All of the user input features of Batcmd (see Chapter 19) were written in Builder. Builder has a number of different ways to get input from the user, including single characters and long strings. You can ask the user a yes/no question, accept only numbers, letters or function keys, and much more.
- **Math.** While batch files have no mathematical abilities, Builder can perform a full set of mathematical and logical operations on integers between plus and minus two billion.
- **String operations.** Builder includes a full set of string manipulation and string logic testing functions. You can easily find the length of a string, convert it to all lower- or uppercase, or pick out three characters in the middle of a string.
- **No more spaghetti code.** Builder supports structured programming methods like Case, While, Repeat and If-Then-Else.
- **Subroutines.** Subroutines can be built into a program, or they can be written as stand-alone programs and attached to the main program while compiling.

This is only a very partial list. Builder offers so much that it can do anything that you'd ever want from a batch file!

Speed of a compiled language

Batch files generally run slow because DOS reads them a little at a time and then has to pause and figure out what to do, all because batch files are written in instructions that a human (and not a computer) can understand. However, when the computer reads a program like the ones produced by Builder, it reads the entire program into memory so that it does not have to access the disk again for that program. Additionally, the compiler converts your program into instructions that the computer can understand, thus not forcing the computer to perform any translations when running the program. This results in a much faster program.

Full compatibility with the MS-DOS batch subcommands

Builder will read in any existing batch file and convert it automatically to a Builder program. Because of all the enhancements Builder offers, you will be able to do away with almost all of your other batch file utilities.

Ease of learning of the batch language

I could write pages and pages about how easy it is to learn to program in Builder, but you probably would not believe me. Rather than doing that, I'm going to show you the source code to a Builder program; I'm willing to bet that you'll understand most of the program, without having any prior knowledge of the Builder language.

The program shown next is CAPITAL.EXE; it takes a single input from the user, converts it to all uppercase, and stores the results in the environment under the environmental variable name RONNY. Additionally, it has built-in documentation, offers command line help if you start it with a /? or just a question mark, and

performs extensive error-checking to make sure that it was passed exactly one word to convert to uppercase. Knowing just that, does this code still make sense to you?

```
'CAPITAL.BLD
'Convert A Single Input To Uppercase
'Store In The Environment Under The Name Ronny
'1.00
'May 16, 1992
STRING Input1
IF (("%1"=="/?") OR ("%1"=="?"))
 SAY "Converts A Single Word Input To Uppercase"
 SAY "And Stores The Results In The Environment ";
 SAY "Under The Name Ronny"
 SAY "Sets Errorlevel To 1 If There Is A Problem"
 EXIT 0
END
IF "%1"==""
 SAY "Nothing To Convert"
 BEEP
 EXIT 1
END
IF NOT ("%2"=="")
 SAY "Only One Word Can Be Converted But Both ";
 SAY "<YELLOW>%1 And %2<NORMAL> Were Entered"
 BEEP
 EXIT 1
END
Input1 := "%1"
Input1 := UPPERCASE Input1
SET "Ronny" = Input1
EXIT 0
```

Most of this code is devoted to documentation, help and error-checking. The Builder code to just convert a string to uppercase and store the results in the environment—without everything else is

```
STRING Input1
Input1 := "%1"
Input1 := UPPERCASE Input1
SET "Ronny" = Input1
```

The first line declares a string variable that the program needs for this example. The second line stores the first replaceable parameter into this variable. The third line converts this variable to uppercase and stores the results back to the same variable name, and the last line stores the results in the environment.

Notice: So easy, and it even looks like a batch file. Builder does it all!

Builder Lite

If Builder sounds like an excellent idea to you, you will find a coupon in the back of this book for a discount on the program. If you are looking for something to improve your batch files but Builder sounds like more than what you need, TAB Books has a

special deal: we have worked with hyperkinetix to take the Builder features most beginners want and have combined them into a less powerful program called Builder Lite.

Builder Lite is a book/software combination available from TAB. In the book, I explain how to use all the features of Builder Lite with lots of examples. This is a good way to get started. Once you have mastered Builder Lite, if you feel like you need even more power, you can upgrade to Builder.

Summary

Distributing batch files to other users is complicated because they might have a different brand or version of operating system than you do. Also, they probably will have different batch file utilities than you do.

Builder programs have all of the Builder batch language enhancements compiled into the program, so anyone who receives a copy of your Builder program has access to all the enhancements you used.

Builder offers a wealth of enhancements to the batch language such as menus, user input, math, string operation, structured code, subroutines, and much more.

Builder programs run much faster than batch files.

Builder is fully compatible with all DOS batch commands.

If you already know batch programming, then the Builder language is easy to learn.

Builder Lite offers the most popular features of Builder at a lower price.

Part six

Batch file applications

We've learned so much about batch files, and now it's time to put that into practice. So far, you've seen a number of batch files that solve very specific problems. For the next few chapters, we will look at how batch files can be a part of the solution for larger problems.

An automated batch file help system One problem with having a lot of batch files is remembering which batch file does what. As it turns out, batch files can solve this problem and, in the process, give you a set of very powerful tools for locating the right batch file for your needs. **Chapter 28**

Simple batch file menus The most common thing you do with your computer is run different programs. Often, this involves entering several different commands. Well, batch files can automate the process and make it simple for you to start your different applications. **Chapter 29**

A document archival system Never lose an important file again! That is a strong claim, but this chapter will show you how. **Chapter 30**

Simulating Doskey using batch files Doskey gives you command line recall, keyboard macros, and the ability to enter multiple commands on a single command line. As it turns out, all these features can be added to DOS using nothing more than batch files and this chapter shows how. It even turns out that there are significant advantages to using batch files over Doskey. **Chapter 31**

Smaller batch file applications This chapter presents a number of batch file applications that were too small to put in a separate chapter. **Chapter 32**

Applications and problems Think of this as a final exam. You get to try out everything you have learned in this book so far. **Chapter 33**

An automated batch file help system

One problem with having a lot of batch files is that it is difficult to remember which batch file does what. One way around this is with a menu. Menus are nothing more than a list of available options along with some method of selecting one of those options. Menus are discussed in detail in the next chapter. However, if you write a menu system with an entry for every one of your batch files, you are going to have to work through a lot of menus to find a specific batch file.

Remember: One approach to providing batch file help is a menu. ☑

Help files

Another approach is to write your own help file, like HELP.BAT shown below without documentation and command line help (the disk version has both):

```
@ECHO OFF
CLS
ECHO DISCARD.BAT
ECHO -----------
ECHO Will Move All Unwanted Files
ECHO To A Directory For Holding
ECHO SYNTAX: DISCARD FILE1 FILE2
BATCMD SL
ECHO MAINTAIN.BAT
ECHO ------------
ECHO Will Erase Temporary Files,
ECHO Sort Files, And Run Your
ECHO File Defragmentation Program
ECHO SYNTAX: MAINTAIN
ECHO WARNING: Takes Two Hours To Run
```

```
BATCMD SL
ECHO PRINT.BAT
ECHO ---------
ECHO Will Print ASCII Files Automatically
ECHO SYNTAX: PRINT FILE1 FILE2 FILE3
ECHO WARNING: Make Sure Printer Is On
ECHO Or Computer Will Lock Up
```

This batch file displays a friendly reminder of the purpose of each batch file.

☑ Remember: A second approach to providing batch file help is a batch file that displays topical information.

As written, HELP.BAT simply displays a brief explanation of each batch file. Once you have accumulated a lot of batch files, it can be time-consuming to read through all those messages looking for a specific batch file. As HELP1.BAT in FIG. 28-1 shows, IF-tests can be used to make this help batch file somewhat context-sensitive. However, this help is limited to the few categories preprogrammed into the batch file.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: HELP1.BAT`
`REM PURPOSE: Display Batch Help`
`REM VERSION: 1.00`
`REM DATE: January 3, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`IF (%1)==() GOTO NOTHING`	If the user did not enter a topic, jump to the appropriate section.
`CAPITAL %1`	Convert the first word of the topic entered by the user to all capital letters. This avoids case problems in IF-tests.

`IF %RONNY%==UTILITY GOTO UTILITY`
`IF %RONNY%==UTILITIES GOTO UTILITY`
`IF %RONNY%==PRINTING GOTO PRINTING`
`IF %RONNY%==PRINT GOTO PRINTING`
`IF %RONNY%==BACKUP GOTO BACKUP`
`IF %RONNY%==BACK GOTO BACKUP`
`IF %RONNY%==MISC GOTO MISC`
	Test the user's input and jump to the appropriate section. Notice the attempt to catch errors by using multiple spellings on some terms.

`GOTO WRONG`	If the batch file reaches this point, the user entered an invalid topic so jump to an error-handling routine.

28-1 *HELP1.BAT adds limited context sensitivity to a batch-based help facility.*

Batch File Line	Explanation
:UTILITY ECHO DISCARD.BAT ECHO ---------- ECHO Will Move All Unwanted Files ECHO To A Directory For Holding ECHO SYNTAX: DISCARD FILE1 FILE2... BATCMD SL ECHO MAINTAIN.BAT ECHO ---------- ECHO Will Erase Temporary Files, Sort ECHO Files, And Run Your ECHO File Defragmentation Program ECHO SYNTAX: MAINTAIN ECHO WARNING: Takes Two Hours To Run GOTO END	Display the help messages for the "utility" topic.
:PRINTING ECHO PRINT.BAT ECHO --------- ECHO Will Print ASCII ECHO Files Automatically ECHO SYNTAX: PRINT FILE1 FILE2 FILE3 ECHO WARNING: Make Sure Printer Is On ECHO Or Computer Will Lock Up GOTO END :BACKUP ECHO BACKUP.BAT ECHO ---------- ECHO Will Backup All (Or Some) ECHO Subdirectories On Your Hard Disk ECHO SYNTAX: BACKUP ECHO To Backup Entire Hard Disk ECHO SYNTAX: BACKUP Directory ECHO To Backup One Subdirectory GOTO END	Display the help messages for the "printing" topic.
:MISC ECHO GAME.BAT ECHO -------- ECHO Will Bring Up The Game Menu ECHO SYNTAX: GAME GOTO END	Display the help messages for the "misc" topic.
:WRONG ECHO INVALID SYNTAX GOTO NOTHING	Section to display an error-message when an invalid topic is entered. Notice that it jumped to another section for more help.
:NOTHING ECHO SYNTAX IS HELP1 CATEGORY ECHO Valid categories are: ECHO Utility, Backup, Printing, Misc GOTO END	Section to display help when the user does not enter a topic.
:HELP ECHO HELP1.BAT Displays Help On ECHO Specific Batch Files GOTO NOTHING	Display a help screen when the user requests help. Notice that it too jumps to another section to display more information.
:END	Label marking the end of the batch file.

28-1 *Continued.*

Customized help

While writing menus or custom help batch files works, it is a difficult system to maintain. Each time you write a batch file or discard an existing batch file, you must update everything to reflect that change. Otherwise, your help system gets out of "sync" with your batch files and gives outdated information.

☠ Warning: While writing menus or topical batch files does work, it is difficult to keep these systems in sync with all your batch files.

One approach allows you to add and delete batch files at will and never gets out of sync; however, it requires some special considerations when you create your batch file. First, it helps if you store all your batch files in a common subdirectory. I use the C:\BAT subdirectory for this, although the name does not matter; if you use a different subdirectory name, you will need to modify these batch files to work with your subdirectory name.

While you don't need to keep all your batch files in one subdirectory, it makes the help system work much better. If you have your batch file arsenal scattered across multiple subdirectories, the help system can only give you information on one subdirectory at a time. Also, for these batch files to work across different subdirectories, you will need to modify them to either take the subdirectory as an input or always use the current subdirectory.

Second, your batch files must be created with the proper internal documentation and must use consistent capitalization, although readers with DOS 5 or later do not have to bother with capitalization because the FIND filter in later versions of DOS will ignore capitalization. This requirement affects only the first five lines of the batch file, and every batch file should begin with these same first five lines:

```
@ECHO OFF
REM NAME:
REM PURPOSE:
REM VERSION:
REM DATE:
```

Chapter 4 examines this in more detail. If you have not read Chapter 4 yet, you should do that before you continue with this chapter.

☑ Remember: The easiest way to display batch file help is using a system that keys off of the information built into the batch file itself.

HELPBAT.BAT in FIG. 28-2 uses this documentation to display custom help about each batch file. It begins with a FOR-loop that loops through each batch file in the C:\BAT subdirectory. For each batch file, it calls itself with a special flag that causes HELPBAT.BAT to jump to a special routine the second time through.

That special subroutine first displays the name of the batch file and underlines it. It has access to the name because the name is passed to it as a replaceable parameter. After displaying the name, it types the batch file using the TYPE command. It pipes this information to the FIND filter, which then displays only the line containing PURPOSE:. This is the reason for the recommendation in Chapter 4 to put your entire purpose statement all on one line; that way, this display completely explains the purpose of the batch file.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: HELPBAT.BAT`
`REM PURPOSE: Display .BAT Purpose`
`REM VERSION: 1.00`
`REM DATE: November 6, 1991`
	Documentation remarks.

`IF (%2)==(DISPLAY) GOTO DISPLAY`	This line checks to see if the second replaceable parameter is a DISPLAY, indicating the batch file is calling itself as a subroutine. If so, it jumps to the subroutine section.
`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`FOR %%J IN (C:\BAT\*.BAT) DO` `  CALL HELPBAT %%J DISPLAY`	When running as the main routine, loop through each batch file in the C:\BAT subdirectory and call this batch file as a subroutine. Pass it the batch file name and a flag to indicate its subroutine status. If you keep batch files in a different location, change the "C:\BAT" to the name of that subdirectory. To work in the current directory, change the first portion of the command to "FOR %%J IN (*.BAT)".
`GOTO END`	After looping through all the batch files in the main routine, exit the batch file.
`:DISPLAY`	Label marking the beginning of the subroutine.
`  ECHO %1`	Display the name of the batch file by echoing the %1 replaceable parameter, which contains the name since it was passed when the batch file called itself as a subroutine.
`  ECHO ----------`	Underline the name of the batch file to set it off from the purpose.

28-2 *HELPBAT.BAT displays the name of each batch file and its purpose line.*

28-2
Continued.

Batch File Line	Explanation
`TYPE %1 | FIND "PURPOSE:"`	Type the batch file and use the Find filter to search for the line containing the phrase "PURPOSE:". In most batch files, this line will be the purpose line; although, a few batch files--like this one--will have "PURPOSE:" as part of a command. When that happens, two lines will be displayed.
`BATCMD SL`	Use the Batcmd utility to display a blank line.
`GOTO END`	The subroutine is finished, so exit the batch file. Control will return to the version of HELPBAT.BAT running as the main routine.

`:HELP`
`  ECHO Reads Batch Files & Displays`
`  ECHO The Name And Purpose Line`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

If you must use multiple lines for your purpose statement, you could start each one with a PURPOSE: to make sure these help batch files display the entire purpose. That does, however, tend to make the help screens look fairly cluttered. After that, control returns to the original version of HELPBAT.BAT that loops through to the next batch file, if any remain.

On a 16-MHz IBM Model 80, this system runs slow enough that I can read all the lines as they scroll up the screen. However, on my Northgate 386/20, the lines scroll too fast to read. If that happens to you, you can replace the `ECHO %1` line with a `TYPE %1 ¦ FIND "NAME:"` line. That will cut the speed of the display in half because the batch file now has to type the batch file twice and use the FIND filter twice to locate specific text. If it still runs too fast, you can add lines to display the version and last modified date. If you display all four pieces of information, that version of HELPBAT.BAT will run at one-fourth the speed of the original version.

When you delete a batch file, it ceases to exist, so the FOR-loop bypasses any reference to it and it is automatically removed from this system. When you add a new batch file, it contains all the information required by HELPBAT.BAT and the FOR-loop automatically includes it, so it is automatically included in this system. As a result, this system is automatically current with your batch files without any modification when you modify your batch files.

Context-sensitive help

While HELPBAT.BAT is a very effective method of displaying general batch file help, it can be overpowering when you have a lot of batch files. For example, I have over 700 batch files for all the books I have written. Finding a specific batch file using HELPBAT.BAT would take a long time and a lot of reading.

Fortunately, narrowing the search is fairly easy. You simply only need to add a filter to the line that types the batch files, as shown by HELPBAT2.BAT. When you do not enter any text, you get all the purpose lines as you do with HELPBAT.BAT. However, when you enter a phrase, HELPBAT2.BAT displays the purpose line for only those batch files that contain that phrase.

HELPBAT2.BAT is very similar to HELPBAT.BAT but it differs in two areas. First, the phrase you enter to search on must be constructed as an environmental variable. This phrase must appear on the purpose line, and it must appear in exactly the same order specified without anything else in the middle. If the purpose line was

```
REM PURPOSE: Runs The Menu Program
```

then the phrases "Menu," "Menu Program," and "The Menu" would all display this line. However the phrases "Run The Menu" (with "Run" instead of "Runs"), "Runs Menu" (without the "The"), and "Menu Program Now" (extra word) would all fail to display this line.

The section of batch file code that constructs the environmental variable is shown below:

```
:TOP
 IF NOT (%FIND%)==() SET FIND=%FIND% %1
 IF (%FIND%)==() SET FIND=%1
 IF NOT (%FIND%)==() SET BLANK=NO
 SHIFT
 IF (%1)==() GOTO RUN
 GOTO TOP
```

This code loops around constructing the FIND environmental variable by adding one "word" at a time until it runs out of replaceable parameters. After that, this is used as an additional search parameter; as shown below:

```
:DISPLAY
 IF %BLANK%==YES GOTO BLANK
 ECHO %1
 ECHO ----------------------
 TYPE %1 ¦ FIND "%FIND%" /I ¦ FIND "PURPOSE:" /I
 BATCMD SL
GOTO END
```

Notice how it skips this additional searching by jumping to a different section if there were no replaceable parameters, but includes a second FIND filter search if replaceable parameters were entered. It is required because using a FIND filter to search on an empty replaceable parameter causes none of the lines to match.

While HELPBAT2.BAT only displays the purpose line for those batch files that contain the text the user specifies on the command line, it displays the underlined file names of every batch file in the subdirectory. This too can be avoided; however, it requires several steps:

1. The results of the FIND search is piped into a file. If the FIND search is successful, this file will contain the "PURPOSE:" line; otherwise, it will be a 0-length empty file.

2. The file (called JUNK.@@@) is checked with IsItZero to see if it is a 0-length file. If so, the batch file currently being processed does not match the search criteria so it is skipped.
3. If the file is not a 0-length file, then the name of the batch file is displayed and the "PURPOSE:" line is displayed by typing JUNK.@@@ to the screen.

HELPBAT3.BAT accomplishes this. First, it searches for the text with the command:

```
TYPE %1¦FIND "%FIND%" /I¦FIND "PURPOSE:"/I>JUNK.@@@
```

This pipes the results of the search to a temporary file. Next, it finds out if this is a 0-length file with the command:

```
ISITZERO JUNK.@@@
IF ERRORLEVEL 4 GOTO END
```

If the batch file makes it past these two lines then the file is not a 0-length file so the search was successful. Finally, it displays the batch file and purpose line with the commands:

```
ECHO %1
ECHO ----------------------
TYPE JUNK.@@@
```

☑ Remember: HELPBAT3.BAT uses the information built into the batch file itself to display help on only those batch files relating to the topic you specify.

Depending on how many batch files you have, you will find either HELPBAT.BAT or HELPBAT3.BAT a big help in figuring out which batch file does what. I know I use HELPBAT3.BAT constantly on my system.

Summary

One way to keep track of all your batch files is to access them through a menu system.

If you track your batch files with a menu system, you must update the menu system every time you add or delete a batch file.

Another way to track your batch files is to write yet another batch file to provide help on existing batch files. This batch file can be non-context-sensitive like HELP.BAT or context-sensitive like HELP1.BAT.

Before you can write a more advanced help system, you must store your batch files in a common subdirectory and construct them with the proper internal documentation.

HELPBAT.BAT automatically displays the purpose line for each batch file. It does not require any modifications when you add or delete batch files.

HELPBAT2.BAT uses the FIND filter to only display the purpose line for batch files that contain a specified piece of text. However, HELPBAT2.BAT still displays the name of all the batch files.

HELPBAT3.BAT uses the FIND filter to only display the purpose line for batch files that contain a specified piece of text. By using DOS-specific syntax, HELPBAT3.BAT avoids displaying the names of batch files that do not match the search criteria.

29 Simple batch file menus

Back in 1983, the organization I worked for got its first few computers. These computers were shared by over twenty users, most of whom didn't know much about computers at that time. After answering the question "How do I start Lotus?" constantly for several weeks, I developed a menu system very much like the first menu system I will describe in this chapter. I placed this menu system on the most frequently used computer. The MIS manager saw it and laughed at it, saying that no one would ever use it. Several days later, I saw him copying it onto the other computers! He thought it wasn't necessary, but the users loved it.

Note: Menus might not seem necessary, but users love them.

The purpose of a menu system is to handle much of the overhead involved in performing routine tasks with a hard disk like changing subdirectories and starting programs. For the most part, menus are more trouble than they're worth for a floppy-disk system. Before developing a menu system, however, you should have a well-structured hard disk; let's spend a few moments examining hard disk structure.

Remember: The purpose of a menu system is to handle much of the overhead involved in performing routine tasks.

Structuring your hard disk

A floppy disk is like a desk. You can just toss all your papers on a desk and usually find what you need. Similarly, you can copy files onto an available floppy disk and likely find the file later, especially if you label the disk.

A hard disk, on the other hand, is like a filing cabinet. If you just toss in all your papers, it's unlikely that you'll ever find anything. An average user will create or accumulate over one thousand files a year. Of course, if you install Windows and a

few Windows-based applications, your total will be higher. If they were all in a single directory, simply getting a listing would take a long time; finding one specific file would be next to impossible. Just as you categorize all the papers in a filing cabinet into files, you should categorize the files on a hard disk into directories and subdirectories.

An additional reason to create these divisions is that DOS places an arbitrary limit on the number of files that can be created in the root directory. Because of the way DOS stores subdirectories, this limit doesn't apply to them.

The first division from the root directory is considered a directory, and further divisions from there are subdirectories. Keep in mind, however, that these two terms are often used interchangeably.

Each subdirectory is stored as a file in the parent directory. Because the subdirectory is a file, it can expand to hold as many files as needed. A subdirectory is a logical subdivision of a disk, much as a folder is a logical division of a filing cabinet. DOS treats each subdirectory much like a separate floppy disk. Each subdirectory has a separate directory listing, and each one can have files with the same name. Subdirectories don't significantly slow down most DOS operations, and they don't take up much room, so there's no real disadvantage to creating as many as you need. Each subdirectory can also grow to fill all the available space on the hard disk, so using subdirectories doesn't make it more difficult to fit files on a hard disk.

With a complex subdirectory structure, it's easy to become "lost" on your hard disk. You can reduce the chance of this happening by including a "pg" at the end of your prompt. The $p displays the full path to the current subdirectory including the drive, so you always know where you are. The $g displays the traditional greater-than sign that terminates most prompts.

☑ Remember: A subdirectory is a logical subdivision of a disk.

While a subdirectory can have as many files as you want, there are technical reasons to limit the number of files in a single subdirectory to a manageable number. DOS tries to hold all the filenames of a subdirectory in memory, which speeds directories and searches. However, when the number of files is too large to fit into memory, DOS slows down dramatically. You'll notice this problem because DOS will have to read the disk more than once when you perform a DIR. At this point, you should either reduce the number of files in the offending subdirectory or increase the number of buffers specified in the CONFIG.SYS file.

The DOS commands for managing subdirectories are CD (CHDIR), FDISK, FORMAT, MD (MKDIR), and RD (RMDIR). These commands perform the following functions:

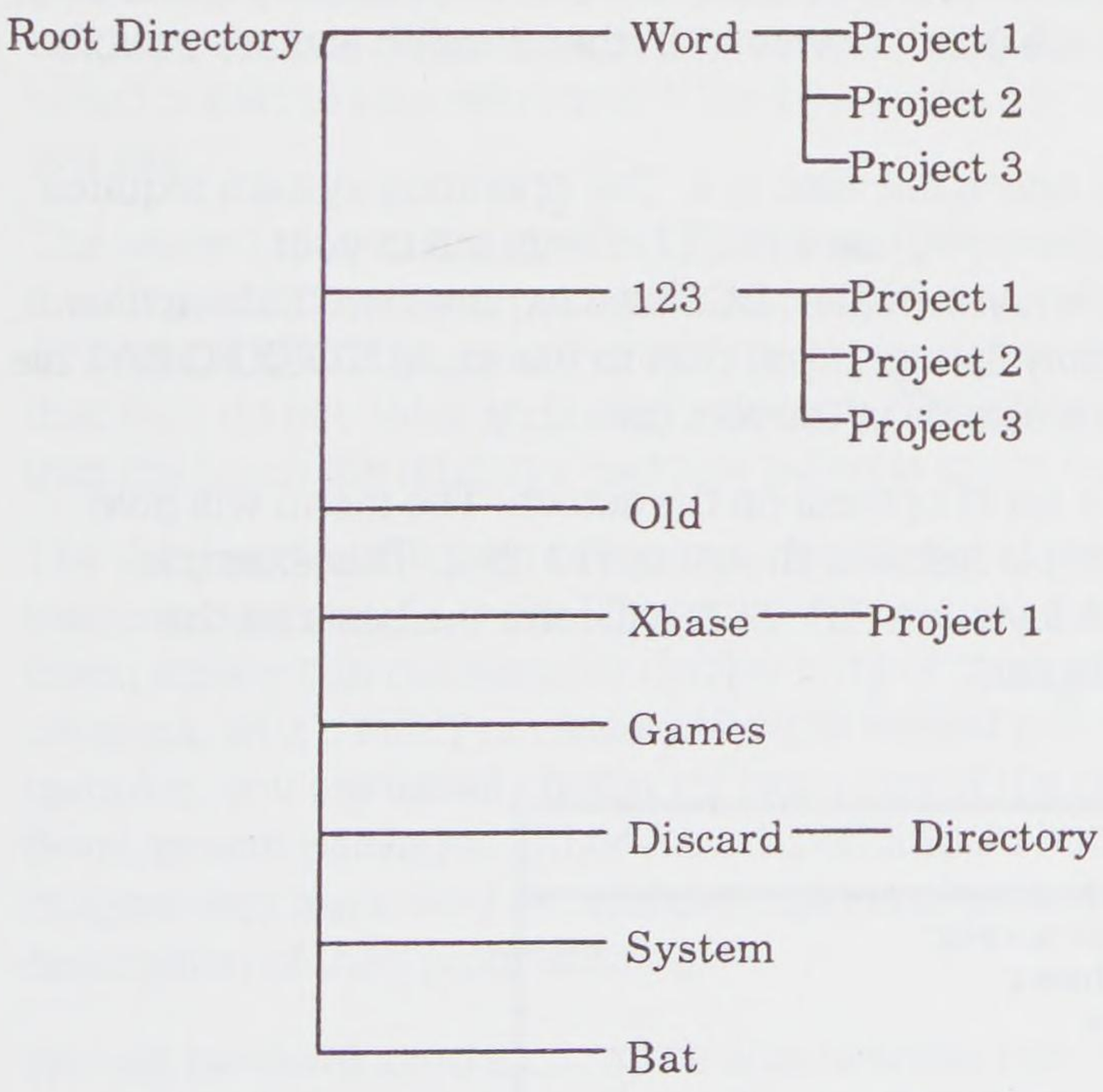
CD *path*	Changes the currently active directory to the one named in the path.
FDISK	Used to prepare a new hard disk. If the disk has any information already on it, FDISK will destroy the information.
FORMAT	Once a new disk has been prepared with FDISK, FORMAT lays down a logical structure on the disk.
MD *name*	Makes a subdirectory with the name you specify that branches off the currently active directory.

`RD` *name* Removes the subdirectory name if it branches off the current directory and if it doesn't contain any files.

Remember: The DOS commands for managing subdirectories are CD, FDISK, FORMAT, MD, and RD.

Think of the subdirectories of a disk as a tree. Each disk has only one root (or base) directory, which is created automatically when you format the disk. The root directory can contain as many subdirectories as you need, although the technical limit is the number of files DOS allows in the root directory. Each of these subdirectories can have subdirectories, and so on. The limit is that the path statement (C:\sub-1\sub-2\etc.) must be less than 63 characters long.

When a directory is created, it will be a child (or branch) of the current directory (called the parent). So if you create a subdirectory while in the root directory, that subdirectory would be a branch off the root directory. If it were created when you're in a subdirectory, it would be a branch of that subdirectory. A drawing of all the subdirectories in a hard disk, therefore, looks a lot like an inverted tree. Figure 29-1 shows a diagram of the subdirectory structure of a typical hard disk.



29-1
A typical hard disk configuration.

Each subdirectory (except the root directory) will automatically contain two files when you create it; these show up as a dot and double dot file when you perform a DIR. These two files have special meanings to DOS and allow for two handy shortcuts. The double dot file stands for the parent subdirectory, so you can move one subdirectory closer to the root directory with the `CD..` command. You can continue issuing this command until you reach the root directory. The single dot file represents all the files in the subdirectory, and can be used in place of `*.*` when copying or erasing files.

To remove a directory, that directory must not have any files except the required dot and double dot files. If you find you can't remove a subdirectory, first check to see if it has any files in it. If so, you must erase them. When DIR doesn't show any files, you most likely have hidden files. The hidden status of the file can be changed with the ATTRIB program, but be sure you know what you are doing because some files—like some of the operating system files—must be hidden files to run.

When you're changing directories, you can move only along a continuous path. For example, in the hard disk diagrammed in FIG. 29-1, you can't move directly from C:\WORD\PROJECT1 to C:\123\PROJECT2. You must first move back to the root directory. However, you can move back to the root directory and on to the desired subdirectory with the single command `CD\123\PROJECT2`. The CD is the change directory command, and specifying the entire path, from just to the right of the C:, moves you back to the root directory and on to the desired subdirectory. `CD\` by itself always returns you to the root directory.

Typically, the first level of subdirectories (or directories) are functional. There is one for system files, one for word processing, one for spreadsheets, and so on. Each of these directories can then have subdirectories. So the word processing directory for a teacher could have subdirectories for personal letters, course outlines, tests, consulting, and so on. If you teach several courses, the test subdirectory could even be subdivided by course. At some point, however, further division is more trouble than it's worth.

The root directory should have only a few files in it. The operating system requires it to have COMMAND.COM unless you use a SHELL command in your CONFIG.SYS file and two hidden system files. DOS also requires two hidden files that you will not see in a directory listing. If you plan to use an AUTOEXEC.BAT file or a CONFIG.SYS file, they must also be in the root directory.

Introduction to menus

A menu is nothing more than a list of options on the screen. The menu will give numbers for each option. A sample menu is shown in FIG. 29-2. This example menu, along with the hard disk layout in FIG. 29-1, will form the basis for the remaining examples in this chapter.

29-2
A sample menu.

Press	For
1	Run Word Processor
2	Run Spreadsheet
3	Run Database
4	Play Chess
5	Format A Floppy Disk
6	Backup Hard Disk To Floppies

Menu options can do more than just automatically run a program; they can also control how dangerous programs are used. For example, if you try to format a floppy disk without the drive specification (FORMAT A:), you could erase your entire hard disk with some versions of MS-DOS. This can be avoided if you format disks from the menu. The menu option to format a disk runs a batch file, thus preventing the user from entering the format command incorrectly.

Displaying the menu

The menu itself is nothing more than text on the screen, and text can be displayed using any of the methods explored in Chapters 20 and 21. For all of the examples in this chapter, I will be using .COM files produced by the BatScreen screen compiler included with this book. Refer to Chapter 20 for documentation on this program.

Types of menus

Menus fall into four very broad categories:

- Non-resident non-nested menus
- Resident non-nested menus
- Non-resident nested menus
- Resident nested menus

The first menu type is called a non-resident menu because, after displaying the menu screen, the menu gets out of memory—leaving you with nothing but DOS. Entering a selection, say 1, at the DOS prompt to run a word processing program with a non-resident menu does nothing by itself; you must also create a batch file called 1.BAT to perform the action under the first option in the menu. Entering 1 runs 1.BAT. In addition to running the word processor, 1.BAT would typically contain a final command to redisplay the menu.

The first menu type was also called non-nested because none of the menu items call other menus. When one of the menu items calls another menu, the menu being called is said to be nested inside the first menu. Menus can be nested as deeply as you like.

The second menu type is called a resident menu because the menu remains in memory, watching over your selection and then executing that selection. The advantage of this approach is that the menu can watch over users and make sure that they do not enter an invalid selection. The disadvantage of this approach is that the batch file driving a resident menu is much more complex.

The third and fourth menu types are duplicates of the first two, only with additional menus nested inside them. These types of menus are much more complex. A single menu screen can comfortably display forty or more options in two or even three columns, so it's rarely necessary to write nested menus. If you need menus this complex, you are usually better off using one of the commercial or shareware menu development packages rather than depending on batch files. For these reasons, this chapter only has a very general description of nested menus and not a full-blown description of their programming.

Non-resident non-nested menus

We will be developing a complete non-resident non-nested menu system as an example in this chapter. All of the files for this demonstration system are stored in the \MENU1 subdirectory on the disk that comes with this book. In order to allow you to experiment with the non-resident non-nested menu system, I've included on the disk slightly altered copies of the batch files shown in the book. These modifications are as follows:

- The batch files always return to the \MENU1 subdirectory rather than the root directory. Because this action only takes place in the MENU.BAT subdirectory, you only need to make this change in one location.

- The batch files do not automatically return to the C drive; instead, they remain on whatever drive you ran them from. Each batch file changes to the drive containing the application it's running, while MENU.BAT takes care of returning to C:\, so each batch file would need to be modified to make the system operational.
- Because there are no applications to run on the floppy disk, the batch files that run the applications merely simulate running them by displaying a screen and pausing until you press a key. Of course, on your own system, you would want to modify each batch file to run your specific applications.

As mentioned earlier, a non-resident non-nested menu is nothing more than a method of displaying the menu screen and a batch file to run each option. The batch file to display the menu screen is called MENU.BAT and is shown next without command line help and without most of its internal documentation:

```
@ECHO OFF
REM C:
REM Change The Line Below To CD\ On Your System
CD\MENU1
SCREEN1
```

As you can see, the only thing it does is change to the appropriate drive and subdirectory and then display the menu via SCREEN1. This runs SCREEN1.COM, a compiled copy of the screen.

☑ Remember: A non-resident non-nested menu is nothing more than a method of displaying the menu screen and a batch file to run each option.

The location of the batch files and screen compiler .COM files that drive this system is important. They must be in a subdirectory that is always in your path. I keep all my files in my C:\BAT batch subdirectory along with all my other batch files.

Now, let's create the first batch file, 1.BAT, to run the word processor program. The entire batch file is shown here because we need to look at it fairly closely:

```
@ECHO OFF
REM NAME:    1.BAT
REM PURPOSE: Run Word Processor
REM VERSION: 1.00
REM DATE:    June 2, 1992
NEEDHELP %0 %1
IF ERRORLEVEL 1 GOTO HELP
REM C:
REM CD\WORD
PROMPT EXIT To Return To Word
REM WORD
REM Delete The Next Two Lines
WORD
PAUSE
PROMPT $P$G
MENU
```

```
:HELP
 ECHO Runs Word Processor
GOTO END
:END
```

After the internal documentation and checks to see if the user requested help, the batch file changes to the drive and subdirectory containing the word processing program, changes the prompt (as mentioned earlier, in order to remind users shelling out of the word processor that it's loaded into memory), and then runs the program. In this batch file, the command WORD runs WORD.COM on the disk—another screen created with BatScreen to simulate a word processor. Users planning on converting this batch file for use on their system should remove this line and the PAUSE command.

After running the applications, the batch file resets the prompt and then runs MENU.BAT to redisplay the menu. However, it runs MENU.BAT without using CALL, so control never returns to 1.BAT. At this point, MENU.BAT picks up, returning to the appropriate subdirectory and redisplaying the menu. The other batch files—2.BAT through 6.BAT—run in a similar fashion. The batch files for formatting a floppy disk and making a backup can easily run from any location on the hard disk, so they skip changing location. Also, because these programs do not allow you to shell out, they also skip changing the prompt.

The real advantage of this sort of system is its flexibility of use. If you want to run the spreadsheet and remember that 2.BAT can do that, you can issue the *2* command anywhere and load the spreadsheet. You don't have to load the menu program first, and you don't have to change to a specific location on the hard disk first. Because 2.BAT is in your path, it works automatically when you issue the *2* command.

Remember: The real advantage of a non-resident non-nested menu is the commands work anytime and anywhere, even if the menu itself is not shown on the screen.

This is the very system I installed back in 1983, only back then I displayed the menu options with ECHO commands in the batch file. That system has been upgraded once to replace the ECHO commands with output from an earlier version of the screen compiler, and it's still in use today. As you can imagine, for an organization to continue using this type of application for almost ten years through three generations of computers means they find it both powerful and easy to use. While I will explain resident non-nested menus in detail and briefly cover nested menus, I'm absolutely convinced that this approach just mentioned creates the best menus you are going to find. It is, in fact, the menu system I use on my personal machines.

Notice: For an organization to continue using this type of application for almost ten years means they find it both powerful and easy to use.

Resident non-nested menus

With a non-resident menu, the batch file displays the menu and then quits running. With a resident menu, the batch file hangs around and waits for the user to press a key. The advantage of this approach is the batch file can watch over users and make sure that they do not push the wrong key. The disadvantage is that writing such a batch file is more difficult, plus the system can be more complex to use.

Because you're trading off extra "hand-holding" against more complex development and use, I only recommend this approach when you are developing a system that will be used by inexperienced users.

☑ Remember: A resident menu remains in memory to validate the user's selection.

You have two approaches when writing resident menus: the hybrid batch file and the stand-alone batch file. With a hybrid batch file, the menu system stays in memory to validate your selection but then calls on stand-alone batch files to run the menu selection. With a stand-alone batch file, a single batch file both validates the user's selection, executes that selection, and then redisplays the menu.

Both of these approaches require something that the non-resident menu does not: an "Exit" or "Quit" command as part of the menu. Occasionally, someone will want to issue a command or run a program that's not on the menu. Because the non-resident menu simply leaves you at the DOS prompt after displaying the menu, that's simple. However, the resident menu keeps you within the cocoon of a batch file, you need a specific command to exit the menu to get to the DOS prompt.

Hybrid batch file

In this section, we will be developing a sample complete hybrid resident non-nested menu system. All of the files for this demonstration system are stored in the \MENU2 subdirectory on the disk that comes with this book. In order to allow you to experiment with the hybrid resident non-nested menu system, the batch files shown in the book have been slightly modified for the disk. These modifications are the same modifications that were mentioned earlier for the non-resident non-nested menu system.

As explained before, the hybrid system uses the same batch files to run the menu selected as does the non-resident system. For that reason, none of the batch files to run the software are printed in this chapter, although all of them are included on the disk.

The difference comes in with MENU.BAT, now called MENU2.BAT. After displaying the menu, it stays in memory waiting for the user to enter a keystroke. When the user makes a selection, MENU2.BAT first verifies that the user made a valid selection. If her solution was valid, it simply runs the proper batch file. Because that batch file will reload MENU2.BAT, it runs the batch file without the CALL command. If the user makes an invalid selection, MENU2.BAT displays an error message and waits for the user to make another selection. MENU2.BAT is shown in detail in FIG. 29-3.

29-3
MENU2.BAT drives a hybrid non-nested menu system.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: MENU2.BAT`
`REM PURPOSE: Display Menu And Position DOS`
`REM In Proper Location & Validate`
`REM User Selections`
`REM VERSION: 1.00`
`REM DATE: June 2, 1992`
	Documentation remarks.

Batch File Line	Explanation
`NEEDHELP %0 %1`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF ERRORLEVEL 1 GOTO HELP`	If the user requested help, jump to a special section to display that information.

`REM C:`
`REM Change Line Below To CD\ On Your System`
`CD\MENU2`
	Change to the appropriate drive and subdirectory on your system.

`SCREEN2`	Display the screen.
`:RETRY`	Label used to loop when the user makes an invalid selection.
`BATCMD GN`	Use the Batcmd command to obtain a number from the user.
`IF ERRORLEVEL 7 GOTO WRONG`	If the user entered 7-9, that is an invalid selection so jump to an error-handling section. Because errorlevel tests are greater-than-or-equal, the possible values from Batcmd (0-9) must be tested from largest to smallest.

`IF ERRORLEVEL 6 6`
`IF ERRORLEVEL 5 5`
`IF ERRORLEVEL 4 4`
`IF ERRORLEVEL 3 3`
`IF ERRORLEVEL 2 2`
`IF ERRORLEVEL 1 1`
	If the user entered 1-6, then they made a valid selection corresponding to a batch file so run that batch file. Since the batch file is run without the Call command, control does not return directly to MENU2.BAT; however, each batch file calls MENU2.BAT as its last command.

`IF ERRORLEVEL 0 GOTO END`	If the user selected zero, they want to exit the menu system to leave the batch file.

29-3
Continued.

29-3
Continued.

Batch File Line	Explanation
:WRONG	Label marking the beginning of the error-handling section. The batch file only reaches this section and beyond via a Goto command.
BATCMD PC 19 1	Position the cursor using the Batcmd program.
ECHO Invalid Selection	Display an error-message.
BATCMD BE	Beep the speaker using the Batcmd program.
GOTO RETRY	Go back through the user selection loop again.
:HELP ECHO Displays The Menu GOTO END	Display a help screen and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.

☑ Remember: A hybrid menu system offers the advantages of a non-resident system plus user selection validation.

As you can see from FIG. 29-3, MENU2.BAT is only somewhat more complex than MENU.BAT shown previously. Additionally, 1.BAT through 6.BAT will still function as stand-alone batch files. So you have the additional error-checking that the resident menus offer, plus the advantage of the batch files working from anywhere without having to run the menu that the non-resident menus offer. Thus, at the expense of a slightly more complex menu batch file, a hybrid menu system offers all of the advantages of both the non-resident and resident systems without the complexities of the full resident menu system. That makes the hybrid system well suited for those applications—like new users—where you want the control of a resident system without the added complexity.

Stand-alone batch file

In this section, we will be developing a sample complete stand-alone resident non-nested menu system. All of the files for this demonstration system are stored in the \MENU3 subdirectory on the disk that comes with this book. In order to allow you to experiment with the stand-alone resident non-nested menu system, the batch files shown in the book have been slightly modified for the disk. These modifications are the same modifications explained for the non-resident non-nested menu system earlier.

☑ Remember: A stand-alone resident menu is one where a single batch file displays the menu, validates selections and runs menu options.

A stand-alone resident non-nested menu is one where a single batch file displays the menu, handles user selection validation, and handles running all of

the applications. These tend to be massive batch files that are difficult to write and harder to maintain. The stand-alone approach is mostly useful under three circumstances:

- **Compilers.** You plan on compiling the batch file using a batch file compiler, like Builder or Builder Lite. However, most batch file compilers offer their own menuing options that are usually much more powerful than those in batch files, so here you are better off using the native language to develop the menu than you are the batch language. A few compilers (like Bat2Exec) offer no native menu support, so you must write them in the batch language.
- **Space.** If your hard disk is very pushed for space, you can often save a little space by using one massive batch file over several smaller ones. However, this approach requires the presence of a utility like Batcmd to get information from the user. If you have that on your hard disk anyway, it's not a consideration. If you do not have Batcmd on your hard disk, then the combination of Batcmd and the large batch file can exceed the space required for the individual batch files.
- **Security.** If you want to control who runs the applications or how they are run, then building that security into a central batch file is much easier than building it into each individual batch file. Additionally, not being able to run individual batch files adds additional security. In fact, security is the very best justification for a stand-alone approach to resident menus.

While the batch file that drives a stand-alone resident menu is long and complex, it possesses one secret that makes it easier to write and debug. This batch file is nothing more than the batch file used for the hybrid menu, only instead of running another batch file for valid selections, it runs one of its own internal routines using the GOTO command.

Remember: Stand-alone resident menus are easier to create and debug if you start by developing a hybrid resident or a non-resident system and merging all the batch files together once it is operational.

MENU3.BAT is the driving menu for my sample stand-alone resident menu. It is shown in FIG. 29-4. To create MENU3.BAT, I took MENU2.BAT (FIG. 29-3) and modified the errorlevel tests to GOTO a separate section of the batch file rather than running a separate batch file. For each of these six sections, I imported 1.BAT through 6.BAT, stripped off the documentation and command line help, and added a GOTO command to the bottom. Once I finished those two steps, MENU3.BAT was finished and ready to use.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: MENU3.BAT`
`REM PURPOSE: Stand-Alone Resident`
`REM Non-Nested Menu`
`REM VERSION: 1.00`
`REM DATE: June 2, 1992`
	Documentation remarks.

`NEEDHELP %0 %1`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.

29-4
MENU3.BAT drives a stand-alone non-nested menu system.

Batch File Line	Explanation
`IF ERRORLEVEL 1 GOTO HELP`	If the user requested help, jump to a special section to display that information.
`:TOP`	Label marking the top of the menu portion of the batch file. As each menu option terminates, it will jump here to redisplay the menu.

`REM C:`
`REM Change Line Below To CD\`
`  On Your System`
`CD\MENU3`
	Change to the appropriate drive and subdirectory on your system.

`SCREEN2`	Display the screen.
`:RETRY`	Label used to loop when the user makes an invalid selection.
`BATCMD GN`	Use the Batcmd command to obtain a number from the user.
`IF ERRORLEVEL 7 GOTO WRONG`	If the user entered 7-9, that is an invalid selection so jump to an error-handling section. Since errorlevel tests are greater-than-or-equal, the possible values from Batcmd (0-9) must be tested from largest to smallest.

`IF ERRORLEVEL 6 GOTO 6`
`IF ERRORLEVEL 5 GOTO 5`
`IF ERRORLEVEL 4 GOTO 4`
`IF ERRORLEVEL 3 GOTO 3`
`IF ERRORLEVEL 2 GOTO 2`
`IF ERRORLEVEL 1 GOTO 1`
	For each of these valid selections, jump to a separate section of the batch file to handle the selection.

`IF ERRORLEVEL 0 GOTO END`	If the user selected zero, they want to exit the menu system so leave the batch file.
`:WRONG`	Label marking the beginning of the error-handling section. The batch file only reaches this section and beyond via a GOTO command.
`  BATCMD PC 19 1`	Position the cursor using the Batcmd program.
`  ECHO Invalid Selection`	Display an error-message.
`  BATCMD BE`	Beep the speaker using the Batcmd program.
`GOTO RETRY`	Go back through the user selection loop again.

`:1`
`  REM C:`
`  REM CD\WORD`
`  PROMPT EXIT To Return To Word`
`  REM WORD`
`  REM Delete The Next Two Lines`
`  WORD`
`  PAUSE`
`  PROMPT $P$G`
`GOTO TOP`
	Section to handle the first menu option. Notice how these commands exactly mirror the commands in 1.BAT. That makes it fairly easy to convert a hybrid system to a stand-alone system.

29-4
Continued.

Batch File Line	Explanation
:2 REM C: REM CD\123 PROMPT EXIT To Return To 1-2-3 REM 123 REM Delete The Next Two Lines 123 PAUSE PROMPT PG GOTO TOP	Section to handle the second menu option.
:3 REM C: REM CD\XBASE PROMPT EXIT To Return To XBASE REM XBASE REM Delete The Next Two Lines XBASE PAUSE PROMPT PG GOTO TOP	Section to handle the third menu option.
:4 REM C: REM CD\GAMES PROMPT EXIT To Return To Chess REM Chess REM Delete The Next Two Lines Chess PAUSE PROMPT PG GOTO TOP	Section to handle the fourth menu option.
:5 FORMAT-1 A: PAUSE GOTO TOP	Section to handle the fifth menu option.
:6 BACKUP-1 C:\ A: /S PAUSE GOTO TOP	Section to handle the sixth menu option.
:HELP ECHO Displays The Menu GOTO END	Display help and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.

29-4
Continued.

Nested menus

A nested menu is nothing more than one menu that has one or more of its menu items as additional menus rather than a program. Nested menus are more difficult to set up using stand-alone or hybrid batch files than they are with stand-alone batch files. The reason is numbering. Suppose the menu we have been working on had option four "Play Chess" replaced by "Play Games," and that menu selection brought up a selection of the different games you have installed on your system. The first menu used options 1–6 corresponding to 1.BAT to 6.BAT, so this second menu cannot easily reuse these numbers. This second menu might use A–G

corresponding to A.BAT to G.BAT. Now, if you add a third and fourth menu, what are you going to use to select the items with?

You might try A–G for the first menu, H–M for the second, and N–S for the third. But what do you do when you need to add a menu item to the first menu? If you make it H, all the menus have to be renumbered and the commands the users have learned no longer work the same. If you use T, because it's the next available letter, then the menu items are not numbered sequentially. There is no good solution to this problem with a stand-alone system.

You have the same problem with a hybrid system if you want the menu options used within the hybrid system to correspond with the batch files—that is, if selecting option 1 runs 1.BAT. If you are willing to give that up and have option 1 run a batch file like WP.BAT, then nesting menus with a hybrid system is no more difficult than it is with a stand-alone system. However, this gives up the advantage of a hybrid system because knowing the menu option used inside the hybrid system does not allow you to run that program from anywhere on the hard disk without working through the menu first.

Warning: Finding unique, single-digit names for each menu option in a nested menu is difficult with non-resident and hybrid systems.

Because everything is built into a stand-alone system, the numbering scheme used for one menu cannot interfere with the number scheme used for another menu, so you are free to reuse numbers or letters at will. That makes developing nested menus much easier in a stand-alone system.

I must warn you that programming nested menus with batch files is difficult; and the more complex the menu structure, the more difficult it is. Your best approach if you have a lot of programs to run is to try and fit all the options on one, or at most two, menus. By using short descriptions and a multi-column format, you can easily list sixty or more menu options on a single screen. By using the twenty-six letters and ten single-digit numbers, the most programs you can run with a single-digit command is thirty-six, and that will easily fit on a two-column screen with room left for instructions and formatting.

If you need more menu items than this, you are probably better off using a dedicated menu system rather than batch files. One of the better dedicated menu systems is Automenu from Magee Enterprises. This is a shareware program, so you'll find it on many bulletin board systems.

Summary

The purpose of a menu system is to handle much of the overhead involved in running programs and other routine tasks.

A hard disk is like a filing cabinet; it needs a consistent filing system.

A subdirectory is a logical subdivision of a hard disk.

The CD or CHDIR command is used to change subdirectories.

The FDISK command is used to prepare a hard disk for use. Using it erases anything stored on the hard disk.

The MD or MKDIR command is used to create a subdirectory.

The RM or RMDIR command is used to remove an existing subdirectory once the files have been erased.

A map of the subdirectories on a hard disk looks like an inverted tree.

First-level subdirectories are typically functional with subdirectories below that for data.

The four broad types of menus are non-resident non-nested, resident non-nested, non-resident nested, and resident nested.

A non-resident menu displays the menu option and then leaves the user at the DOS prompt to make his/her selection.

A resident menu displays the menu option and then remains in memory to validate the user's selection.

In a non-nested menu, none of the menu items calls up an additional menu.

In a nested menu, one or more of the menu items call up additional menus.

A non-resident menu displays a test on the screen and then depends on batch files to run the menu options.

A non-resident non-nested menu allows you run the menu options anytime and anywhere on the hard disk, even if the menu is not being displayed.

Resident menus are more complex than non-resident menus.

A hybrid resident menu turns over the execution of menu options to other batch files after it has validated the selection.

A stand-alone resident menu handles menu display, selection validation, and option execution.

Nested menus are more difficult in non-resident and hybrid systems because of the difficulty of arranging for different batch file names for each menu option.

Nested menus are easier in stand-alone systems, but if you need to nest more than one or two menus, you are probably better off with a dedicated menu system.

30 A document archival system

What I'm going to describe in this chapter is a complete system for making sure that you never again lose an important file. Because it consists of several major sections, you can adopt only those sections that are critical to your own applications. The major sections to my overall document archival system are listed next.

Note: This is a complete system to make sure you never again lose an important data file!

- **Periodic backups.** Absolutely nothing is more important than periodically backing up your hard disk or important floppy disks. This is the only part of this plan that I recommend to everyone. In fact, without periodic backups, the rest of this system is fairly useless. If you don't care enough about your data to protect it with a backup, then why waste time protecting it in any other fashion?
- **Systematically copy critical data files.** Many important data files undergo routine revisions. Prior to making a major revision, I copy the file to be revised to a special holding subdirectory.
- **Never delete old files.** I never erase an old file. Instead, I copy it to a special holding subdirectory. When that subdirectory is full of files, I copy the files to a disk and erase them from the subdirectory. I then store the disk in a safe place. I currently have 125 such disks.
- **Index document files.** I use a commercial indexing package to index my word processing documents so I can find them faster.

Each part of my overall document archival system is explained in much more detail, as follows next.

Periodic backups

Back when everyone used floppies, the most damage you could do with one mistake was 320K. Like mistakes, bad disks or defective hardware rarely damaged more than 320K. Large hard disks, however, now give us the ability to damage 200M or more of data with one mistake or as the result of an errant command or defective hardware. 200M is over 135 high-capacity floppy disks!

Warning: If you don't care enough about your data to back it up, then why waste time protecting it in any other fashion?

I insure my computer with a company called Safeware. Recently, they sent out a newsletter listing some of the more unusual claims they had paid. This list included the following:

- A California man carried some work home with him. He kept the disks in a box that looked like his cat's litter box. As you might expect, the cat used the disk box to relieve himself and destroyed all the data on the disks.
- A California boy was hiding behind the family computer during a friendly game of hide-and-seek. When a friend found him, the boy made a heroic dash to avoid being tagged. While dashing, he tripped over the power cord. That sent a shock through the computer that caused the hard disk to crash.
- A Los Angeles motorist collided with a fire hydrant, sending a column of water into the air. That water entered a nearby office building, damaging several computers.
- A California man knocked his fish tank over, spilling all the water into his computer.
- A Florida man had his laptop out near the pool. The family dog bumped the table and sent the computer to the bottom of the pool.

As you can see, computers can be damaged by a number of unexpected events. While an insurance policy like the one from Safeware can protect your hardware and even your programs, it's up to you to protect your data.

In the last presidential campaign, I saw a graphic reminder of the importance of backups. CNN reported a break-in at the campaign headquarters of Ross Perot. The only damage reported was the erasure of a file containing the names of 17,000 supporters. As you can imagine, this has the potential of doing millions of dollars of damage to a campaign if these 17,000 supporters are the more generous supporters. In a close race, it could even determine the outcome! However, at the close of the report, the CNN reporter noted that the Perot campaign had a backup disk.

Many small businesses maintain all their records—all the payroll records, tax information, bid information, and more—on a single microcomputer. Imagine the impact to that business if those records were lost. I know of one company that had their three computers stolen. To make matters worse, they didn't have backups. They were so desperate for the data that they ran ads saying that the thieves could keep the computers and they wouldn't press charges if they could just get their data back. They never did, and they ended up going out of business.

I know of a large utility company that maintains a lot of their rate case information on microcomputers. One of the analysts tried to format a floppy with (you guessed it) the FORMAT command at the C prompt and ended up reformatting the hard disk. He was lucky: a recent backup had been made and he lost only a few hours of work. The manager told me that losing the files on his hard disk would have delayed a multimillion-dollar rate case. He now requires daily backups.

➠ Note: Losing the files on a hard disk could cost a great deal of money and permanently damage a business.

I have about twenty articles in progress, the final version of my dissertation, a copy of this book, the related batch files, eight other finished books and several books in progress on my hard disk. Until recently, I make a daily incremental backup and two full weekly backups. One backup is stored at home and the other is stored at my office.

Is this carrying things too far? I don't think so. An incremental backup takes about two minutes, and the two full backups take less than half an hour. So I spend about an hour a week on backups, this hour being insurance. It would take an experienced typist over twenty hours to rekey just my dissertation, and it would take me thousands of hours to redo the research if I lost the hard disk copy and couldn't locate a printed copy. Even my notes are on the hard disk. Now add in the time to redo all the other files, and you begin to see why I make backups.

As I said, that was my approach until recently. I now have a very fast tape drive that can perform a full backup on both my hard disks in about twenty minutes, which is so fast that I've forgone incremental backups. I now make two full backups daily (one during supper and one after I quit for the night), and I keep the three latest versions of these two backup sets. One set is stored at home and the other is stored at the office.

Which file to back up?

You should have copies of most of your software on the original distribution disks, so you don't need to back up these files very often. Just keep the originals in a safe place so they can't be damaged. Even in the worst case of fire or theft, you can always replace your software by buying a new copy, so you will not lose your software.

☑ Remember: Software comes with its own backup in the form of the distribution disks, making backing up your software less important.

Except during installation, programs don't often change. The software is "tied together" with system files that include batch files, AUTOEXEC.BAT, CONFIG.SYS, and subdirectories. You can safely go a long time between backups of these files. They generally change only when you install new software or remove existing software.

If damaged, these system files can be replaced from an old backup or the original distribution disks. However, things aren't that simple with your data files. You create them and then usually modify them quite often. It's important, therefore, to back up these files frequently.

☑ Remember: You create your data, so it's important to back up these files frequently.

Types of backup

Your first hard-drive backup should be a full backup. This gives you a good copy of installed software and system files, as well as a copy of your data files. When you change a data file, DOS automatically changes a flag it maintains to indicate that a current backup no longer exists. At the end of the day (or week, if you want to live dangerously), you can make an incremental backup. The backup software looks through your hard disk for files with the archive flag. The software backs up that file and resets the flag, and then looks for the next file.

This process is fairly painless because incremental backups are very quick. As you continue making incremental backups, the disks will begin to pile up. If you're constantly updating a 300K database, then you'll create a new copy of this file on disk every day. Most backup software won't remove earlier versions of this file from the backup disks during an incremental backup. Another problem is that files you've erased from the hard disk still exist on the backup disks. Files that have been renamed will exist on the backup disks under both names. If you have to restore from backup disks, you can end up with a lot of extra files.

You can reduce these minor problems by using LISTFILE.BAT. LISTFILE.BAT is shown here without command line help or internal documentation but the version on the disk has both.

```
@ECHO OFF
C:
CD\
ATTRIB *.* /S > C:\CHANGED.TXT
```

All LISTFILE.BAT does is create a file containing a list of all the files on the hard disk. On my 386/20 with a bulging 105M hard disk, it takes less than two minutes to run. To avoid tying up your computer, you can easily run LISTFILE.BAT while running an errand or taking a break.

By running it before backing up, this file will also be backed up—even by an incremental backup because you just changed it. If you must restore, you can compare the files that exist after the restore to LISTING.TXT. Any files that exist on the hard disk but not in LISTING.TXT were either erased or renamed, so they can be deleted.

While incremental backups are quick, restoring from them can be cumbersome if they have been going on too long. At some point you'll have so many disks from incremental backups that it will be time to erase them and make another full backup.

These types of backups will protect you from hardware problems and mistakes but not from theft, fire, or other types of damage. For that, you need a second backup set, stored off-site. In order to create a second incremental backup, your backup program must give you the option of not resetting the archive flag, and not all of them do.

Warning: Frequent backups alone will not protect your data against fire or theft. For that, you need an off-site backup.

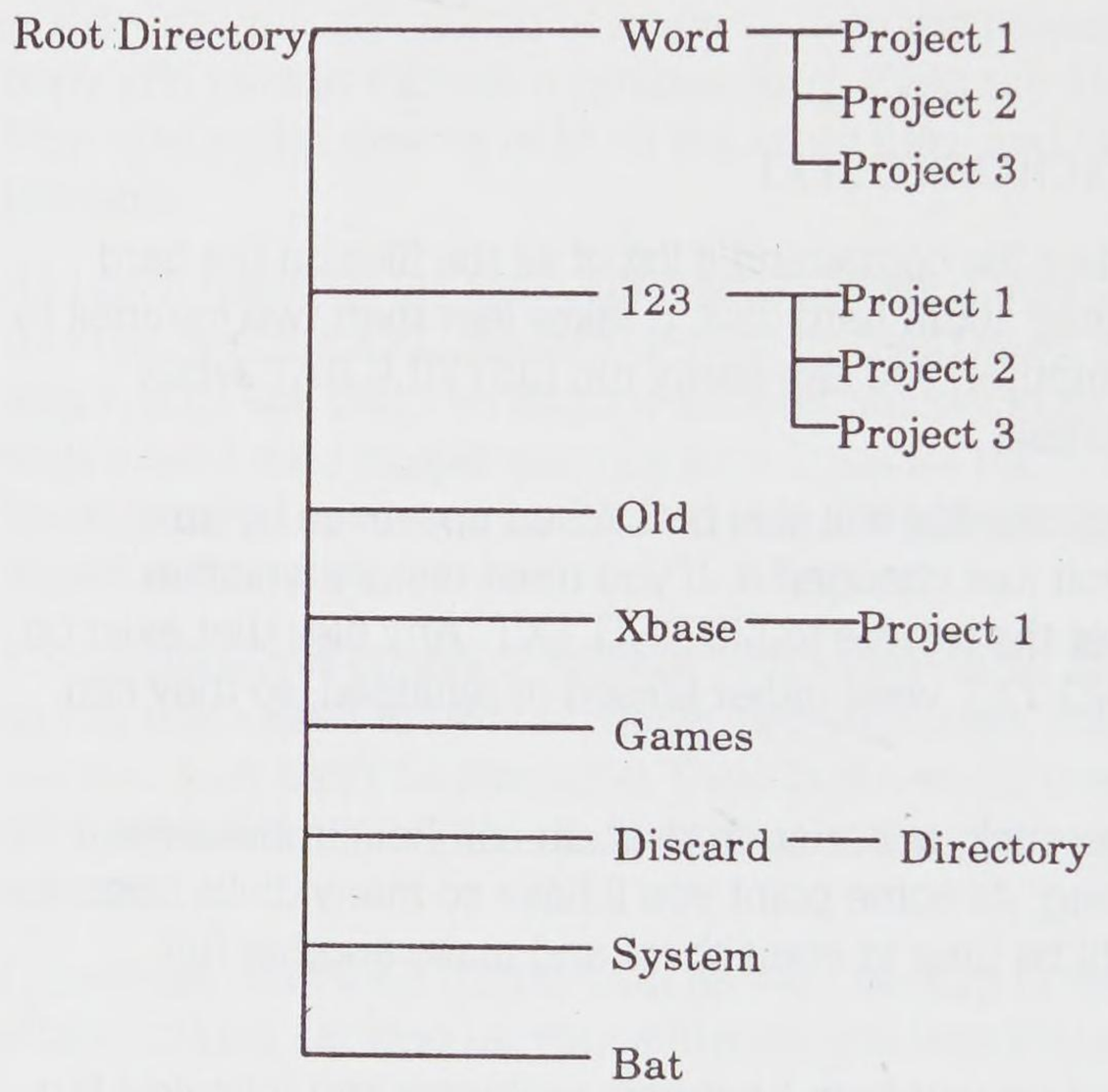
Many organizations with very critical files maintain their three or so newest incremental backups. These are often called the grandfather, father, and son. The purpose of these multiple backups is that you can recreate changes to important files.

Multiple backups also give you a way to recover if your primary backup fails. I once needed to restore from a backup made by a major backup program. That program had not displayed any error messages when making the backup but when I went to restore, not a single file on the backup was recoverable. I sent the diskettes to the software company and they could not recover any data either. Luckily, I had a full backup from the day before, and it restored properly, so all I lost was one day's work. However, without that second backup, all of my data would have been lost.

Systematically copy critical data files

If you need multiple backups for only a few files, there's a better way than running your backup program several times a day. I have a subdirectory called C:\OLD just for old copies of files. Every time I make a major change to a file, I copy it to this subdirectory with a number extension. So the first version of this chapter is CHAP-30.001, the second is CHAP-30.002, and so on. This way, I can always go back to an earlier version if I decide I don't like the changes I've made.

Figure 30-1 shows the layout of the example hard disk for this chapter. Figure 30-2 shows COPYBACK.BAT, a batch file to automate the copying of important files to the backup subdirectory. Note that it checks to make sure that the target filename doesn't exist before copying.



30-1
The hard disk configuration used by the batch files in this chapter.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: COPYBACK.BAT`
`REM PURPOSE: Copy Files To Backup Directory`
`REM VERSION: 1.00`
`REM DATE: June 3, 1992`
	Documentation remarks.

`NEEDHELP %0 %1`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF ERRORLEVEL 1 GOTO HELP`	If the user requested help, jump to a special section to display that information.

30-2
COPYBACK.BAT makes a copy of important files in a backup subdirectory.

Batch File Line	Explanation
`IF (%1)==( ) GOTO ERROR` `IF (%2)==( ) GOTO ERROR`	If the user did not enter the source or target file names, jump to an error-handling section.
`IF EXIST C:\OLD\%2 GOTO EXISTS`	If the target file name already exists, jump to an error-handling section.
`COPY %1 C:\OLD\%2/V` `GOTO END`	Perform the copy and exit the batch file.

`:HELP`
`  ECHO Copies The File Specified As %%1`
`  ECHO To The File Name Specified As %%2`
`  ECHO In The C:\OLD Subdirectory`
`  ECHO Will Warn You If File Already Exists`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:ERROR`
`  ECHO Source Or Target Name Not Entered`
`  ECHO At The Command Prompt Enter:`
`  ECHO COPYBACK SOURCE TARGET`
`GOTO END`
	This is an error section that displays an error-message and exits when the user does not enter the source and target file names.

`:EXISTS`
`  ECHO %2 Already Exists!`
`  ECHO Select Another Name And Try Again`
`GOTO END`
	This is an error section that displays an error-message and exits when the target name entered by the user already exists.

`:END`	Label marking the end of the batch file.

30-2
Continued.

As these files continue to accumulate, you have a choice. You can erase older versions of the now modified (several times over) files, or you can copy them to floppy disks.

Never delete old files!

One of the things I learned a long time ago is that, just as soon as you erase a file, you end up needing it. As a result, I never completely erase a file unless I can easily recreate it. There are a lot of files on your hard disk that you can easily recreate, such as these:

- **.BAK files.** With most word processors, every time you save a file, the prior version is saved under the same name with a .BAK extension. Because I save all critical versions to a special subdirectory (as I explained previously), I routinely erase these files.
- **.OBJ files.** When you run most compilers, they first turn your source code into .OBJ files, and then they turn the .OBJ files into .EXE files. If you have the source code, you can easily recreate the .OBJ files by running the compiler again.
- **.PRN files.** These are ASCII files created by printing a spreadsheet file to a document. These can easily be recreated by reprinting the spreadsheet.

- **.$$$ and .TMP files.** These are working files created by some commercial packages. They're usually erased by the package when you exit, but a reboot or disk problem can leave them on your hard disk.
- **Commercial program files.** These are the files that come on your software disks. If you need to recreate these, you can copy them from the original disks.

These are about all the files I erase. For the rest of my files, when I'm finished using them, I copy them to a holding subdirectory called C:\DISCARD, and then I erase the original. Figure 30-3 shows DISCARD.BAT, the batch file I use for this. It copies all the files I list on the command line. Prior to copying, it checks to make sure that a file doesn't already exist in the \DISCARD subdirectory with the same name. After successfully copying the file, it erases the original.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: DISCARD.BAT`
`REM PURPOSE: Moves Files To \DISCARD`
`REM VERSION: 1.00`
`REM DATE: June 3, 1992`
	Documentation remarks.

`NEEDHELP %0 %1`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF ERRORLEVEL 1 GOTO HELP`	If the user requested help, jump to a special section to display that information.
`:TOP`	Label marking the top of the main loop.
`IF (%1)==() GOTO END`	Once there are no more replaceable parameters, exit the batch file.
`IF EXIST C:\DISCARD\%1 GOTO ERROR`	If the file to be discarded already exists in the discard subdirectory, jump to an error-handling routine.
`COPY %1 C:\DISCARD\%1 /V`	If it reaches here, the file is not in the discard directory so copy it there.
`IF EXIST C:\DISCARD\%1 DEL %1`	If the copy was successful, erase the original file.
`SHIFT`	Move all the replaceable parameters down one level.
`GOTO TOP`	Continue looping.

`:ERROR`
`ECHO %1 Already Exists In C:\DISCARD`
`ECHO You Must Decide Which File To Keep`
`GOTO END`
	Display an error-message and exit when the file already exists in C:\DISCARD.

`:HELP`
`ECHO Moves Files To \DISCARD`
`ECHO List Files To Move After Name`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

30-3
DISCARD.BAT is a batch file to copy files to a discard subdirectory.

Once I get a lot of files in the C:\DISCARD subdirectory, I copy them to a floppy disk and then delete them from the hard disk. Each of my floppy disks is numbered, from one to my latest one. After copying the files to a floppy disk, I perform three steps. First I erase the copied files from the hard disk. Next, I create a list of the files on the floppy disk. This list stays on the hard disk for easy retrieval when I want to search for a file. Finally, I index the document files on the hard disk.

Deleting all the files on the hard disk that were copied to the discard floppy disk would be easy if all the files were copied—all it would take is a `DEL *.*` command. However, I sometimes have more files than will fit on a single floppy, but not enough to justify using two. The problem is then to delete only those files that were actually copied to the discard floppy disk. DELOLD.BAT does just that. While it contains a number of lines to handle command line help and internal documentation, the operational commands are

```
A:
FOR %%j IN (*.*) DO DEL C:\DISCARD\%%j
C:
```

The batch file changes to the A drive first because it would be more difficult to write this batch file to run from the C drive.

To create a file on the hard disk containing a listing of all the files on the floppy disk, I change over to the C:\DIRECTORY subdirectory and issue the command

```
DIR A: > 99.DIR
```

replacing the 99 with the number of the current floppy disk. By maintaining a directory of all my discard floppy disks on my hard disk, I can rapidly search them for a file.

This might sound like overkill, especially if you rarely go back to a file after you're finished with it. The advantage to this system is that it's dirt cheap. In my area, 1.44M floppy disks currently cost about $10 per box of ten. So I have less than $125 invested in storing every one of my old files. The way I figure it, using just one of those files again saves me much more than $125. I've been using this system for several years now, and I'm very pleased with the results. In fact, I find these disks so useful, I recently started using DISKCOPY to make a duplicate to keep off-site for safety.

Note: I've been using this system for several years now and I'm very pleased with the results.

Currently, a number of good word processing indexing programs are available. These programs take a word processing file or an ASCII file and create an index of it. These aren't indexes like you find in the back of a book; rather, they're indexes that allow you to search for every document that contains a specific phrase.

Index document files

ZyIndex is such a package. It will index most word processing documents, using the format they're stored in. All you have to do is tell ZyIndex the format. I use ZyIndex to index all the text files on my discard floppy disks so I can rapidly locate information on them. A big drawback to ZyIndex, and all other indexing programs, is that it will index only text information; it won't index dBASE database files or Lotus spreadsheet files. To get around that problem, I create a small text file with the same name but a different extension that explains what the attached file does.

For example, I create .WKT files to explain worksheet (spreadsheet) files. So HP.WKT explains in ASCII text what the file HP.WK1 does. I use .DBT files to explain the function of .DBF dBASE database files.

ZyIndex lets you assign names up to 30 characters to the floppy disks. These names can't include spaces, so I use an underscore to simulate a space. Because all the floppy disks are numbered, I give them the same name combined with their unique number. Disk number 43 would be named DISCARD_DISK_43.

INDEX.BAT runs the indexing program. It is shown below without the command line help or internal documentation of the version on the disk:

```
@ECHO OFF
IF (%1)==() GOTO NONE
CD\ZYINDEX
IF EXIST A:*.DOC ZYINDEX D:\ZYINDEX\ A:*.DOC/03
DISCARD_DISK_%1
IF EXIST A:*.WKT ZYINDEX D:\ZYINDEX\ A:*.WKT/01
DISCARD_DISK_%1
IF EXIST A:*.DBT ZYINDEX D:\ZYINDEX\ A:*.DBT/01
DISCARD_DISK_%1
GOTO END
:NONE
 ECHO You Did Not Enter A Number
 ECHO For The Floppy Disk
GOTO END
:END
```

It knows that the name should include the DISCARD_DISK_ specification, so I don't have to enter that. It also knows a number should be entered, so it checks to make sure that one was.

Summary

By following all four steps outlined in this chapter, you can be sure that you will never lose an important data file. Additionally, you will have the tools and techniques to quickly locate any single data file in your set of floppy disks that you use to store the old data files. I would be interested in hearing from any users who have success with this system or who find ways to improve this system.

The major steps to this plan are periodic backups, duplicating critical data files before modification, copying old files to floppy disks rather than erasing them, and indexing data files with a commercial indexing program.

Periodic backups are critical to the health of your data.

Regular full backups and daily incremental backups will ensure the safety of your data.

For further protection, you must keep rolling backups where you keep the last two or three backups.

For full protection, you need to keep both on-site and off-site copies of all your backups.

Program files and the batch files that support the operation of your system need backing up far less frequently than your data files.

LISTFILE.BAT makes sure every incremental backup includes a list of all the files currently on the hard disk.

If you make a backup copy of critical data files before modifying them by copying the file to a new name in a backup subdirectory, you can always revert to the prior version if you decide that you do not like the changes you made.

COPYBACK.BAT automates the copying of important files to the backup subdirectory so you can keep multiple copies of those important files as you make modifications to them. This allows you to revert to earlier versions as needed.

For additional protection, as this subdirectory of prior versions of critical files grows, it can be copied to floppies rather than simply deleting files.

Program files can be safely deleted because you can restore them from the distribution diskettes as required.

Data files that are easily recreated, such as .PRN and .OBJ, can safely be deleted as required.

All other data files should be moved to a holding subdirectory rather than being deleted.

As this holding subdirectory fills up or when you need additional space on the hard disk, the files can be copied to floppy disks and then deleted from the hard disk.

A list of files for each floppy disk can be maintained on the hard disk to make locating a specific file quicker.

A commercial program can index the data files on the floppy disk to speed file location even more.

31 Simulating DOSKEY using batch files

MS-DOS 5 comes with a wonderful keyboard enhancement program called DOSKEY, as does MS DOS 6. This memory resident program gives you three major command line enhancements:

- Command line macros
- Recall of prior command lines
- The ability to enter multiple commands on a single command line

When I've talked to MS-DOS users who have switched to DR DOS, far and away the most common complaint I hear is about having to give up DOSKEY. I have also heard complaints from some network users that DOSKEY is incompatible with a few network drivers. Finally, DOSKEY is a memory-resident program that significantly reduces available memory—something some of you cannot afford.

Well, I have a solution for you. In this chapter, you will see how to write batch files that give you most of the functionality of DOSKEY without any memory-resident component. In some cases, the batch files will even outperform DOSKEY!

Command line macros

With a command line macro, you enter a macro name and a command or series of commands on the command line. Unlike a command line recorder, a macro is not run immediately after it is created—it's merely stored in memory for later use. After that, every time you enter the macro name at the start of a command line, DOSKEY replaces it with the command assigned to it.

With DOSKEY, these assignments are stored in memory. That means they are lost when you turn the computer off or reboot. While you can save them to a file, you must edit that file before you have a way of recalling the definitions by using the file.

In order to recreate command line macros with a batch file, the batch file will have to be able to do the following things:

- **Store a new macro.** It can do this by piping the commands entered on the command line to a batch file.
- **Run a macro stored in a file.** It can do this by running the batch file containing the macro.
- **Display the available macros.** It can do this by simply typing the contents of each macro to the screen.

MACRO.BAT in FIG. 31-1 does all this. It restricts you to the macro names of 0–9, but clever users can modify this easily. It also limits your commands to nine "words" because %1, which contains the name of the macro, is shifted down to %0. However, clever users can easily modify this as well.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: MACRO.BAT`
`REM PURPOSE: Run Macros From File`
`REM VERSION: 1.00`
`REM DATE: November 6, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`IF NOT EXIST C:\MACRO\NUL MD C:\MACRO`	If the subdirectory used by this batch file to store macros does not exist, create it.
`IF (%1)==() GOTO MISSING`	If the user failed to tell the batch file what to do, jump to an error-handling section.
`IF (%1)==(D) GOTO DISPLAY` `IF (%1)==(d) GOTO DISPLAY`	If the user requested a listing of the available macros, jump to a section to handle that.
`IF (%1)==(ABC123XYZ) GOTO SHOWTHEM`	If the first replaceable parameter is a subroutine flag, jump to that section.
`IF (%2)==() GOTO RUN`	If second replaceable parameter is blank, the user wants to run a macro so jump to that section.
`FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO` `  IF (%1)==(%%J) GOTO DEFINE`	At this point, the user wants to define a macro. Loop through the acceptable names and jump to that routine if the user entered a valid name.

31-1
MACRO.BAT allows you to define, view, and run keyboard macros.

Batch File Line	Explanation
`GOTO ERROR1`	At this point, the user entered an invalid macro name so jump to an error-handling routine.

`:ERROR1`
`  ECHO You Entered An Invalid Code On`
`  ECHO The Command Line`
`  ECHO`
`GOTO MISSING`
	When the user enters an invalid macro name, display an error-message then jump to another section.

`:ERROR2`
`  ECHO The Command You Entered Was Too`
`  ECHO Long And Everything After The`
`  ECHO %9 Is Missing From The Macro`
`  ECHO`
`  ECHO Try To Shorten Your Command`
`GOTO END`
	When the user tries to create a macro with more than nine "words", display a warning then exit the batch file.

`:ERROR3`
`  ECHO The Macro You Selected (%1) Does`
`  ECHO Not Exist`
`  ECHO`
`GOTO MISSING`
	When the user tries to run a macro that does not exist, warn the user then exit the batch file.

`:RUN` `  IF NOT EXIST C:\MACRO\%1.BAT GOTO ERROR3`	Make sure the requested macro exists. If not, jump to an error-handling routine.
`  C:\MACRO\%1.BAT`	Run the macro. Without the Call command, control does not return to this batch file.
`GOTO END`	Control does not return so this is used only for formatting to close the loop.

`:MISSING`
`  ECHO To Define A Macro, Enter The Number`
`  ECHO Followed By The Macro Defination`
`  ECHO`
`  ECHO Numbers 0-9 Are Allowed`
`  ECHO`
`  ECHO To Run An Existing Macro, Enter`
`  ECHO The Number Of That Macro`
`  ECHO`
`  ECHO The Available Macros Are:`
`GOTO DISPLAY`
	When a required value is missing, display an error-message and exit the batch file.

`:DISPLAY`
`  FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO`
`  CALL MACRO ABC123XYZ %%J`
`GOTO END`
	Loop through the ten possible macro names and for each one, call MACRO.BAT as a subroutine, passing it the batch file name then exit.

31-1
Continued.

Batch File Line	Explanation
:SHOWTHEM IF NOT EXIST C:\MACRO\%2.BAT GOTO END	If this particular macro does not exist, exit the subroutine.
ECHO --%2--	Display the macro name.
TYPE C:\MACRO\%2.BAT|FIND /V "@ECHO OFF" ECHO GOTO END	Type the batch file and display the line without the "@ECHO OFF" followed by a blank line, then exit subroutine.
:DEFINE SHIFT	Move all replaceable parameters down one level. This makes the macro name %0.
ECHO @ECHO OFF > C:\MACRO\%0.BAT	Pipe an "@ECHO OFF" to the macro.
ECHO %1 %2 %3 %4 %5 %6 %7 %8 %9 >> C:\MACRO\%0.BAT	Pipe the commands entered on the command line to the macro.
SHIFT	Move all the replaceable parameters down one level.
IF NOT (%9)==() GOTO ERROR2	If %9 still exists, then something was left out of the macro so jump to an error-handling section.
GOTO END	Exit the batch file.
:HELP ECHO MACRO.BAT Allows You To Define ECHO Up To Ten Macros (0-9) That Are ECHO Stored On Disk So They Do Not ECHO Use Any Memory GOTO END	Display a help screen and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.

31-1
Continued.

If you enter MACRO D, it displays all the available macros. If you enter MACRO #, it runs the macro stored under the number you enter on the command line. And if you enter MACRO # *commands*, it stores those commands in a macro without executing them. Macros are stored in the C:\MACRO subdirectory. The subdirectory should not be in your path. They remain until you overwrite them or manually erase them so macros are retained between sessions.

Users interested in implementing this system but concerned about just being allowed to enter one line of commands should note that once MACRO.BAT has created a macro, nothing prevents you from going behind it and enhancing the macro batch files it creates because they are nothing but regular batch files. Additionally, there is nothing preventing you from creating one or more of the

macro batch files yourself without using MACRO.BAT. You will still be able to use MACRO.BAT to display and run the resulting batch file.

Command line recorder

I really dislike the way DOSKEY and every other keyboard program I've seen stores old command lines: *Every* single command line gets stored. I tend to enter a lot of short commands, like "DIR" or "D:," and I prefer not to have to search through all these looking for one or two long commands. That is the nice thing about a batch file approach: only those command lines you enter via the batch file are stored, so you control which ones are stored. Enter the command normally and it bypasses the macro program, which is just perfect for short commands. Enter the command after the batch file name and the command is executed, plus it's stored for later use, which is just perfect for longer commands.

Like the macro batch file, old command lines are kept in batch files so they are not affected by rebooting or turning the computer off. In addition, for especially complex command lines, you can edit the batch file directly.

In fact, the command line recorder is very similar to the macro recorder. It does have three differences:

- After storing a command line, that command line is immediately executed.
- You assign macro names but command lines are stored in the next available name.
- After all the names have been used, the oldest command line is discarded and the remaining ones moved down one name to make room for the new command line to be stored.

As it turns out, all three of the differences are easy to accomplish with a batch file. To run a command line after storing it only requires one additional line in the batch file—a line to run the just-created batch file. Assigning the name to the command line only requires searching for the first available name and, if none is found, deleting 0.BAT, renaming the remaining batch files to a number one lower, and then using the name of 9.BAT to store the current command line. Beyond this, the batch file, called CL.BAT, is very similar to MACRO.BAT explained earlier. CL.BAT is reviewed in detail in FIG. 31-2.

31-2
CL.BAT stores, displays, and reruns command lines.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: CL.BAT`
`REM PURPOSE: Store Command Lines`
`REM Using Files Not Memory`
`REM VERSION: 1.00`
`REM DATE: November 6, 1991`
	Documentation remarks.

`IF (%1)==(ABC123XYZ) GOTO SHOWTHEM`	If it is being called as a subroutine, jump to the subroutine portion of the batch file.
`SET USE=`	Reset the environmental variable used by the batch file.

Batch File Line	Explanation
`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==(D)  GOTO DISPLAY` `IF (%1)==(d)  GOTO DISPLAY`	If the user requested the display of the available command lines, jump to a section to handle that.
`IF NOT EXIST C:\CL\*.* MD C:\CL`	If the subdirectory to store command lines does not exist, create it.
`FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO` `  IF (%1)==(%%J) GOTO RUN`	Loop through the ten available macros to see if the user has requested running one. If so, jump to a section to run it.
`FOR %%J IN (9 8 7 6 5 4 3 2 1 0) DO` `  IF NOT EXIST C:\CL\%%J.BAT SET USE=%%J`	At this point, the user is entering a command line to store and run so find the first available slot.
`IF NOT (%USE%)==() GOTO USE`	If a slot was found, jump to the section that processes the command.
`GOTO ALLUSED`	If no slot was available, jump to a section to create one.

`:RUN`
`  C:\CL\%1.BAT`
`GOTO END`
	Section that runs the command line selected by the user. Since the Call command is not used, the GOTO command is used for formatting and is never executed.

`:USE`	Label marking the beginning of the section that stores a command line and then runs it.
`  SHIFT`	Move all the replaceable parameters down one level.
`  ECHO @ECHO OFF > C:\CL\%USE%.BAT`	Pipe an "@ECHO OFF" to the batch file that was selected to store the command line.
`  ECHO %0 %1 %2 %3 %4 %5 %6 %6 %7 %8` `  %9 >> C:\CL\%USE%.BAT`	Pipe the commands to the batch file that was selected to store the command line.
`  C:\CL\%USE%.BAT`	After creating the batch file to store the command line, run it.

31-2
Continued.

31-2
Continued.

Batch File Line	Explanation
`GOTO END`	Since the batch file was run without the Call command, control never returns to this batch file so this command is only used for formatting.

`:ALLUSED`
`  DEL C:\CL\0.BAT`
`  REN C:\CL\1.BAT 0.BAT`
`  REN C:\CL\2.BAT 1.BAT`
`  REN C:\CL\3.BAT 2.BAT`
`  REN C:\CL\4.BAT 3.BAT`
`  REN C:\CL\5.BAT 4.BAT`
`  REN C:\CL\6.BAT 5.BAT`
`  REN C:\CL\7.BAT 6.BAT`
`  REN C:\CL\8.BAT 7.BAT`
`  REN C:\CL\9.BAT 8.BAT`
`  SET USE=9`
`GOTO USE`
	When all the available slots have been used, erase the oldest command line file, move the remaining ones down one level and use the last slot to store the current command line.

`:DISPLAY`
`  FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO`
`  CALL CL ABC123XYZ %%J`
`GOTO END`
	When user requests displaying the command lines, loop through and display them by calling CL.BAT as a subroutine.

`:SHOWTHEM` `  IF NOT EXIST C:\CL\%2.BAT GOTO END`	If this particular command line does not exist, exit the batch file subroutine.
`  ECHO --%2--`	Display the command line name.
`  TYPE C:\CL\%2.BAT | FIND /V "@ECHO OFF"`	Display the contents of the batch file.
`  ECHO`	Display a blank line by echoing an Alt-255.
`GOTO END`	Exit the batch file.

`:MISSING`
`  ECHO You Did Not Enter Anything`
`  ECHO After The Name So CL Does`
`  ECHO Not Know What To Do!`
`  ECHO`
`GOTO EXPLAIN`
	Display an error-message when the user does not tell the batch file what to do and jump to another section.

`:EXPLAIN`
`  ECHO Enter CL D To Display The`
`  ECHO Existing Command Lines`
`  ECHO`
`  ECHO Enter CL # To Reuse One Of`
`  ECHO The Existing Command Lines`
`  ECHO Where # Is The Number Of`
`  ECHO The Command Line To Reuse`
`  ECHO`
`GOTO DISPLAY`
	Explain what the batch file does then jump to another section to display additional information.

Batch File Line	Explanation
:HELP ECHO Enter CL D To Display The ECHO Existing Command Lines ECHO ECHO Enter CL # To Reuse One Of ECHO The Existing Command Lines ECHO Where # Is The Number Of ECHO The Command Line To Reuse GOTO END	Display a help screen and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.

31-2
Continued.

Running multiple commands

MULTI.BAT in FIG. 31-3 allows you to enter as many commands as you like on a command line, up to the limit of 127 characters. It constructs an enviromental variable containing the command by looping through the replaceable parameter and adding each one to this environmental variable until it reaches a replaceable parameter that is a caret (^); the caret tells it that the command is complete. At that point, it executes that command and then resets the environmental variable and starts over. Because replaceable parameters must be separated by spaces, the caret must have a space on each side.

Batch File Line	Explanation
@ECHO OFF	Turn command-echoing off.
REM NAME: MULTI.BAT REM PURPOSE: Issue Multiple DR DOS REM Commands On a Single Line REM VERSION: 1.10 REM DATE: May 7, 1991	Documentation remarks.
NEEDHELP %0 %1 IF ERRORLEVEL 1 GOTO HELP	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
SET COMMAND=	Reset the environmental variable used by this batch file.
IF (%1)==() GOTO ERROR	If the user did not tell the batch file what to do, jump to an error-handling routine.
IF (%1)==(^) GOTO ERROR	If the first replaceable parameter is a caret, the user specified the command wrong so jump to an error-handling routine.
:TOP	Label marking the top of the main loop.
SET COMMAND=%COMMAND% %1	Add the current word on to the end of the environmental variable that contains the command to run.

31-3
MULTI.BAT lets you enter multiple commands on a single command line.

Batch File Line	Explanation
SHIFT	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
IF (%1)==() GOTO RUNLAST	If there are no more replaceable parameter then the batch file has reached the end of the last command so jump to a section to run it.
IF (%1)==(^) GOTO RUN	If the next replaceable parameter is a caret, then the batch file has reached the end of a command with another one following so jump to another section to run this command.
GOTO TOP	Continue looping.
:RUN	Beginning of the section that runs a command with other commands pending.
SHIFT	Shift to move the caret that separated the commands out of the %1 position.
CALL %COMMAND%	Run the command. When the command is a program, the Call command is ignored but when the command is a batch file, the Call command allows this batch file to regain control.
SET COMMAND=	Reset the environmental variable that contains the command now that the command has been executed.
GOTO TOP	Continue looping.
:RUNLAST	Label marking the beginning of the section that handles running the last command.
%COMMAND%	Run the last command. Since the Call command is not used, control does not return to this batch file if the command is a batch file; however, control does return if the command is a program.
GOTO END	Exit the batch file.

31-3
Continued.

Batch File Line	Explanation
:ERROR ECHO Invalid Commands Specified ECHO The Command Is Multi Followed ECHO By Command Lines ECHO Each Line Must Be Separated By ^ PAUSE GOTO END	Display an error message and exit the batch file.
:HELP ECHO MULTI.BAT Runs Multiple ECHO Commands Entered On The ECHO Command Line ECHO ---------- ECHO To Run, Enter: ECHO MULTI Command ^ Command ECHO Where The Commands Are ECHO Separated By A ^ With ECHO Spaces Around It GOTO END	Display a help screen and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.
SET COMMAND=	Reset the environmental variable used by the batch file before exiting.

31-3
Continued.

The caret was used because it is not likely to be used on the command line and because there is no lowercase version of it to complicate testing. Readers needing to use the caret in commands while surrounded by spaces can substitute another unused symbol.

MULTI.BAT uses one nifty trick and has one limitation you need to be aware of. It executes all of its commands using the command:

```
CALL %COMMAND%
```

MULTI.BAT must do this so it can regain control if the command you enter executes a batch file. Interestingly, the CALL has no impact if the command is not a batch file, so "CALL DIR" and "DIR" perform exactly the same function.

Its one limitation is piping. DOS processes pipes before handing processing over to MULTI.BAT, so the pipes will never even get to MULTI.BAT. As a result, none of the commands you execute using MULTI.BAT can use pipes. Otherwise, commands execute the same under MULTI.BAT as they do when run from the command line.

Summary

Between MACRO.BAT, CL.BAT and MULTI.BAT, you have most of the functions of MS DOS DOSKEY program.

With MACRO.BAT and CL.BAT, you also have the advantage of storing information between sessions automatically without your having to do anything special.

32 Smaller batch file applications

This chapter presents a number of shorter batch file applications, some of them mundane and others unique. This diversity has two purposes. First, I want you to see how you can use fairly simple batch files to solve everyday problems. Second, by presenting you with batch files that are used for unusual purposes, I hope to make you think about how you can solve your own unique problems with a batch file.

Maintaining your C:\BAT subdirectory

Several places in this book I've recommended you keep all your batch files in the C:\BAT subdirectory. In fact, the help system covered in Chapter 28 requires this to work properly. However, it can be a pain to create and modify batch files "on-the-fly" because you first have to change to the C drive and then the \BAT subdirectory. EDITBAT.BAT automates editing batch files in the C:\BAT subdirectory without your having to leave your current subdirectory!

While the batch file contains a number of commands for error checking and help, its single operational line is:

```
EDIT C:\BAT\%1.BAT
```

Because this command automatically creates or edits the file in the C:\BAT subdirectory, you do not have to change there first. EDIT does not create a .BAK backup file for you, but you could automate that by adding the following line before the EDIT line:

```
IF EXIST C:\BAT\%1.BAT COPY C:\BAT\%1.BAT C:\BAT\%1.BAK
```

Screen saver

A screen saver is a program that blanks the screen until you press a key. Screen savers are currently the "rage"; you can buy them with all kinds of pictures that are displayed on the screen instead of a completely blank screen. You can also

construct your own screen saver with a very short batch file. NOBURN.BAT does just this. While it has lines for command line help, its two operational lines are:

```
CLS
PAUSE > NUL
```

The CLS blanks the screen and the "PAUSE > NUL" keeps the prompt from reappearing until you press a key. While NOBURN.BAT is not as fancy as the commercial programs, it is free and requires no memory when you are not using it.

Is Windows running?

Certain commands, like CHKDSK/F, should not be issued while Windows is running. One way to find out is with a utility like the IsItWin utility covered in Chapter 26. However, with a few batch commands you can avoid the need for an extra utility. When Windows 3.1 is running, it creates an environmental variable called "windir" only it creates it differently than does DOS. For any other environmental variable, DOS capitalizes the name of the environmental variable but Windows creates windir as a lowercase environmental variable. The contents of windir is the location of the Windows program.

Microsoft's reason for having Windows create windir as a lowercase environmental variable was to keep you from altering its contents and pointing Windows to the wrong subdirectory. Because it's lowercase, the command `ECHO %windir%` will not display its contents and the command `SET windir=new` will create a whole new environmental variable called WINDIR while leaving windir unaltered.

Nevertheless, your batch file can find out if windir exists! CHECKWIN.BAT on the disk does just that. The operational commands for CHECKWIN.BAT follow:

```
SET ¦ FIND "windir" > JUNK.1
COPY JUNK.1 JUNK.2  > NUL
IF EXIST JUNK.2 ECHO Windows Running
IF NOT EXIST JUNK.2 ECHO Windows Not Running
```

Basically, it uses the FIND filter to search for a line containing "windir" and pipes that line to a file named JUNK.1. If Windows is running, that file will contain information; otherwise, it will be a 0-length file. It then copies JUNK.1 to JUNK.2 since the file will only copy if it is not a 0-length file and Windows is therefore running. Note that this batch file will not run under DR DOS because DR DOS copies 0-length files.

Changing subdirectories

Many users have a very complex directory structure that's nested several levels deep. They often need to leave one subdirectory to check something in another subdirectory and then switch back, and are looking for a way to automate this.

You can actually attack this problem in a couple of different ways. The best way is probably using a quick directory changing program like the NCD program in the Norton Utilities or the shareware QC (Quick Change) program from Steven Flores. You can find QC on many bulletin board services, like CompuServe or PC-Link. You can also get it from many local users groups. If you want to order it directly, send $20 to

Steven Flores
11711 E. 27th St.
Tulsa, OK 74129

Be sure to tell Steven where you heard about his program.

If you need or want to tackle the problem with batch files, you have a couple of alternatives. If you're always switching between just a couple of subdirectories, you can write a custom batch file to change to each subdirectory. For example, you could write a batch file called 0-9.BAT with the single command

```
CD\D\LOTUSFIG\0-9
```

to change to that subdirectory. I use this method to change back to my home directory on the network at the office, because entering `CD\D\USERS\MKTEVAL\RICHARDS` is a bit much.

Of course, this can become a complex solution if you have a lot of subdirectories to change between. You can write a batch file to partially solve this problem. This batch file marks the current directory as home and builds a second batch file to always return you to the home directory. The next time you run this batch file, it creates a new home directory and a new batch file to change you to there.

The batch file RETURN.BAT

```
COPY C:\BAT\RETURN C:\BAT\RETURNTO.BAT
CD >> C:\BAT\RETURNTO.BAT
```

copies a file called RETURN to the batch file named RETURNTO.BAT. RETURN contains the single line

```
CD ^z
```

where ^z is an end-of-file marker. Because there's no return in the file, when the next line of RETURN.BAT pipes in the subdirectory, it gets added to the same line as the CD. The result is a line similar to this:

```
CD C:\D\LOTUSFIG\0-9
```

Even though this path includes the drive, this approach will not change drives under most DOS versions. That is, if you are trying to change to the "\D\LOTUSFIG\0-9" subdirectory on the C drive but issue this command from the D drive, DOS will remain logged onto the D drive and will not change to the appropriate subdirectory.

Because the batch file must be in a subdirectory in the path in order for it to work, RETURN.BAT is configured to always use the C:\BAT subdirectory. For that reason, every time you run RETURN.BAT, it overwrites RETURNTO.BAT with a new version. You can expand RETURN.BAT to handle multiple subdirectories by using a replaceable parameter on the command line and creating RETURN1.BAT, RETURN2.BAT, and so on. I'll leave this up to you.

If you don't want to create a custom batch file, you can use the batch file RETURN2.BAT as follows:

```
@ECHO OFF
CD %HOME%
```

There's no way to pipe the current subdirectory into the environment, so this batch file requires you to manually issue the command

```
SET HOME=\D\LOTUSFIG\0-9
```

to tell RETURN2.BAT which subdirectory is home. The only real drawback to RETURN2.BAT is that you have to manually type the home subdirectory into the environment. This method too has trouble changing drives under most DOS versions. You can avoid having to enter the home subdirectory into the environment, as shown in RETURN3.BAT in FIG. 32-1.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: RETURN3.BAT`
`REM PURPOSE: Change & Return`
`REM To A Subdirectory`
`REM VERSION: 1.00`
`REM DATE: April 15, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO GOHOME`	Jump to one section if the user did not enter a subdirectory.
`CD\%1`	If the batch file reaches this point, the user entered a subdirectory to change to so change to that subdirectory.
`SET HOME=%1`	Store the home subdirectory in the environment.
`GOTO END`	Exit the batch file.
`:GOHOME`	Label marking the top of the section to handle the user running the batch file without a replaceable parameter.
`CD %HOME%`	Change to the subdirectory stored in the environment.
`GOTO END`	Exit the batch file.

`:HELP`
`ECHO RETURN3 Subdirectory`
`ECHO Stores That Subdirectory`
`ECHO In The Environment And`
`ECHO Changes To It`
`BATCMD SL`
`ECHO Entering RETURN3 By Itself`
`ECHO Changes Back To The Last`
`ECHO Subdirectory Stored In`
`ECHO The Environment`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

32-1
RETURN3.BAT records your home directory into the environment when you originally change to it, allowing you to automatically change to it later.

Using RETURN3.BAT to change to a subdirectory originally (which you must do anyway) records that subdirectory into the environment automatically. When you use RETURN3.BAT without a subdirectory, it changes back to the last subdirectory you changed to, using RETURN3.BAT with a subdirectory name. If you like, you can modify RETURN3.BAT to record multiple home subdirectories in the environment.

If you routinely change between only a few subdirectories, it is possible to duplicate the power of the Norton Utilities NCD program with a batch file. RCD.BAT in FIG. 32-2 illustrates this. You run RCD.BAT by including the nickname of a subdirectory to change to after the batch file name. RCD.BAT uses a `GOTO %1` to jump to that label where it issues a command to change to that subdirectory. Because you can have multiple labels together in a batch file, it's possible to assign multiple shortcut names to a single subdirectory as RCD.BAT illustrates. The problem with RCD.BAT is that you must manually append each new subdirectory. Additionally, if you specify a shortcut name that does not exist, the batch file aborts with a DOS "Label Not Found" error message. Unlike the methods explained earlier, RCD.BAT has no trouble changing drives because you can easily program that into the batch file.

32-2
RCD.BAT allows you to quickly change subdirectories.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: RCD.BAT`
`REM PURPOSE: Ronny's Change Directory`
`REM VERSION: 1.00`
`REM DATE: November 15, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.

`ECHO If You See The Error Message`
`ECHO Label Not Found`
`ECHO Then You Selected A Subdirectory`
`ECHO Not Stored In RCD.BAT`
	Since the batch file will abort if the user entered an invalid label, these messages warn the user about the problem.

`GOTO %1`	Jump to the label issued as the first replaceable parameter.

`:BAT`
`  C:`
`  CD\BAT`
`GOTO END`
	If the first replaceable parameter was Bat, change to C:\BAT subdirectory and exit the batch file.

`:SYSLIB`
`:DOS`
`  C:`
`  CD\SYSLIB`
`GOTO END`
	If the first replaceable parameter was Syslib or Dos, change to C:\SYSLIB subdirectory and exit the batch file.

`:BOOKBAT`
`  D:`
`  CD\BATFIG\BAT`
`GOTO END`
	If the first replaceable parameter was Bookbat, change to D:\BOOKBAT subdirectory and exit the batch file.

`The user should add their own shortcut names and subdirectories to this batch file in this area before using it.`	

`:HELP`
`  ECHO RCD Nickname`
`  ECHO Changes To That Subdirectory`
`  BATCMD SL`
`  ECHO RCD.BAT Must Be Preconfigured`
`  ECHO To Use The Nickname`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

Running inflexible programs

At an office I used to work at, we had one computer that we used for all our communications. One of the things we did was log onto a remote database that supplied us with custom communications software. Because of the unique features of this database, we had to use their software. One limitation of this software was that it stored the account number, password, and other information in a file called MENU.INF. We had two people who used this database, and they both had different account numbers, passwords, and so on. This communications software could not handle multiple users. The vendor suggested keeping two versions in two different subdirectories; however, we used batch files as a better solution.

The first user ran the configuration program to create MENU.INF with his information. We then copied MENU.INF to a file called RONNY.INF. The second user ran the configuration program, and we once again copied MENU.INF to a file—this time DAVID.INF.

This database software required just this one special file—MENU.INI—for each user. We created several versions of that special file under different names. STARTDAT.BAT, the batch file in FIG. 32-3, copies the appropriate one to MENU.INF.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: STARTDATA.BAT`
`REM PURPOSE: Allow Two Users To`
`REM Access Database`
`REM VERSION: 1.00`
`REM DATE: April 15, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO NOTHING`	If the user did not enter his name, jump to an error-handling routine.
`CAPITAL %1`	Use the Capital utility to convert this value to upper-case and store it in the environment under the variable name Ronny.
`IF %RONNY%==DAVID GOTO OK` `IF %RONNY%==RONNY GOTO OK`	If the user entered a valid name, jump to a section to run the database program.
`GOTO ERROR`	If the batch file reached this point, jump to an error-handling routine.

`:NOTHING`
`ECHO Enter Your Name After`
`ECHO The STARTDAT Command`
`GOTO END`
	When the user does not enter a name, display an error-message and exit.

`:ERROR`
`ECHO Invalid User Name`
`ECHO Try Again Or See Manager`
`GOTO END`
	When the user enters an invalid name, display an error-message and exit.

32-3
STARTDAT.BAT activates different setups for a database program.

Batch File Line	Explanation
:OK	Label marking the beginning of the section to run the database program when a valid name is entered.
COPY %1.INF MENU.INF	Copy the appropriate configuration file to the file MENU.INI, the name used by the database.
MENU /1200/5551212	Start the database, giving it the baud rate and phone number.
GOTO END	After running the database, exit the batch file.
:HELP ECHO This Batch File Allows Ronny ECHO Or David To Access Remote ECHO Database BATCMD SL ECHO Enter STARTDAT RONNY/DAVID ECHO To Start The Database GOTO END	Display a help screen and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.

32-3
Continued.

This same technique can be used with any program that stores the default values in a special file. For example, Microsoft Word stores the document you're working on, your place, and several optional settings in a file called MW.INI. Figure 32-4 shows a batch file called STARTWOR.BAT that Ronny and David could both use to maintain their own versions of MW.INI. There's only one major difference between the batch files in FIG. 32-3 and FIG. 32-4. Because Microsoft Word allows you to change the defaults while running the program, each user's defaults are copied back to the holding file when that user exits Word. The /L is required to force Word to use some of the defaults from the MW.INI file.

Batch File Line	Explanation
@ECHO OFF	Turn command-echoing off.
REM NAME: STARTWORD.BAT REM PURPOSE: Allow Two Users To Store REM Custom Word Configuration REM VERSION: 1.00 REM DATE: April 15, 1991	Documentation remarks.
NEEDHELP %0 %1 IF ERRORLEVEL 1 GOTO HELP	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
IF (%1)==() GOTO NOTHING	If the user did not enter his name, jump to an error-handling routine.
CAPITAL %1	Use the Capital utility to convert this value to upper-case and store it in the environment under the variable name Ronny.

32-4
STARTWOR.BAT is a word processing version of STARTDAT.BAT, the batch file in Fig. 32-3.

Batch File Line	Explanation
`IF %RONNY%==DAVID GOTO OK` `IF %RONNY%==RONNY GOTO OK`	If the user entered a valid name, jump to a section to run Word.
`GOTO ERROR`	If the batch file reached this point, jump to an error-handling routine.

`:NOTHING`
`  ECHO Enter Your Name After`
`  ECHO The STARTWOR Command`
`GOTO END`
	When the user does not enter a name, display an error-message and exit.

`:ERROR`
`  ECHO Invalid User Name.`
`  ECHO Try Again Or See System Manager`
`GOTO END`
	When the user enters an invalid name, display an error-message and exit.

`:OK`	Label marking the beginning of the section to run Word when a valid name is entered.
`  CD\WORD`	Change to the Word subdirectory.
`  COPY %1.INI MW.INI`	Copy the appropriate configuration file to the file MW.INI, the name used by Word.
`  WORD/L %2`	Run Word.
`  COPY MW.INI %1.INI`	Since Word changes the configuration file each time it runs, copy this version back to the file name used to store the configuration for this user.
`GOTO END`	Exit the batch file.

`:HELP`
`  ECHO This Batch File Allows Ronny`
`  ECHO Or David To Use Microsoft Word`
`  BATCMD SL`
`  ECHO Enter STARTWOR RONNY/DAVID`
`  ECHO To Start Program`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

32-4
Continued.

Several readers of prior versions of this book have told me that they use this same method with Ventura Publishing by using multiple copies of its VP.INI file. However, rather than using this trick for multiple users, they report using it to store multiple configurations of Ventura for a single user.

Finding files

While the batch file is not as flexible as some of the commercial programs, you can use it as a fast file-searching program. The DOS ATTRIB (short for Attribute) program was designed to change the attributes of files. It works across subdirectories using a /S switch. Because running ATTRIB without any switches to change the status of files causes it to list all the files it finds matching your file specification, ATTRIB can work as a file finding utility. Figure 32-5 shows FASTFIND.BAT, the resulting batch file, and explains how it operates.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: FASTFIND.BAT`
`REM PURPOSE: Find A File`
`REM VERSION: 2.00`
`REM DATE: April 15, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO NOFILE`	If the user did not enter the name of a file to search file, jump to an error-handling routine.
`GOTO FILE`	If the batch file reaches this point, the user entered a file to search for so jump to a section to do that.

`:NOFILE`
`ECHO No File Entered On Command Line`
`ECHO Syntax is C>FASTFIND file`
`ECHO Where file is the File You`
`ECHO Wish to Find`
`GOTO END`
	The user did not enter the name of a file to search for so display an error-message and exit the batch file.

`:FILE`
`ATTRIB \%1 /S | MORE`
`GOTO END`
	When the user enters a file name, use the Attrib command to search for that file or files and use the More filter in case the number of files exceeds a screen full. After doing that, exit the batch file.

`:HELP`
`ECHO Entering: FASTFIND File`
`ECHO Will Search The Entire Hard`
`ECHO Disk Looking For That File`
`ECHO Wildcards Are Allowed`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

32-5
FASTFIND.BAT uses ATTRIB to create a fast file-searching program.

Running commands occasionally

If you use your computer to store important data, you should make daily incremental backups. If your data is less important, you may decide to forgo daily backups in favor of less frequent backups. The problem with those is remembering to do them. One approach is to occasionally back up the computer.

Normally, you think of batch files as being either the AUTOEXEC.BAT file, which is run every time you turn your computer on, or a stand-alone batch file that's run only when you enter its name. However, there are some things you want your computer to do only occasionally, like make a backup.

OCCASION.BAT in FIG. 32-6 illustrates this concept. To run this from your AUTOEXEC.BAT file, you'll need a CALL OCCASION.BAT statement. This batch file maintains a counter in the form of a 0-length file. When the batch file first starts, it creates COUNT.00. The next time it runs, it renames this file COUNT.01, the next time it renames it COUNT.02, and so on. When it encounters COUNT.09, it renames it COUNT.00 and performs the task it's supposed to perform occasionally—the backup, in this example. That gives this batch file a period of ten.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: OCCASION.BAT`
`REM PURPOSE: Run Programs Occasionally`
`REM VERSION: 1.00`
`REM DATE: April 15, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`FOR %%J IN (00 01 02 03 04 05 06 07 08 09) DO IF EXIST COUNT.%%J GOTO %%J`	Loop through ten values. If a file named Count exists with that extension, jump to a section corresponding to the extension.
`GOTO NOFILE`	If the batch file reaches this point, Count does not exist with the ten acceptable extensions so jump to a section to create it.

`:00`
`REM All Work Done Here`
`BACKUP C:\ A:`
`REN COUNT.00 COUNT.01`
`GOTO END`
	When the file COUNT.00 exists, perform the backup and rename the file to COUNT.01 to increment the counter.

`:01`
`REN COUNT.01 COUNT.02`
`GOTO END`
	Increment the counter and exit the batch file.

`:02`
`REN COUNT.02 COUNT.03`
`GOTO END`
	Increment the counter and exit the batch file.

`:03`
`REN COUNT.03 COUNT.04`
`GOTO END`
	Increment the counter and exit the batch file.

`:04`
`REN COUNT.04 COUNT.05`
`GOTO END`
	Increment the counter and exit the batch file.

`:05`
`REN COUNT.05 COUNT.06`
`GOTO END`
	Increment the counter and exit the batch file.

`:06`
`REN COUNT.06 COUNT.07`
`GOTO END`
	Increment the counter and exit the batch file.

`:07`
`REN COUNT.07 COUNT.08`
`GOTO END`
	Increment the counter and exit the batch file.

`:08`
`REN COUNT.08 COUNT.09`
`GOTO END`
	Increment the counter and exit the batch file.

32-6
OCCASION.BAT backs up the hard disk once every ten times the computer is rebooted.

32-6
Continued.

Batch File Line	Explanation

`:09`
`  REN COUNT.09 COUNT.00`
`GOTO END`
	Increment the counter and exit the batch file.

`:NOFILE`
`  REM Restore Counter File`
`  REM Then Restart Process`
`  TYPE NOFILE > COUNT.00`
`GOTO 00`
	When the counter file does not exist, create it by typing a file that does not exist and piping the results to the counter file. After this, jump to the section that performs the backup.

`:HELP`
`  ECHO OCCASION.BAT Demonstrates How`
`  ECHO Commands Can Run Occasionally`
`  ECHO By Performing A Backup Every`
`  ECHO Tenth Time It Is Run`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

The batch file counts reboots, not days. If you're working with problem software that frequently locks up, therefore, it could end up running backups several times a day. If you leave your computer on for days at a time or use it infrequently, ten reboots could end up being several weeks.

OCCASION.BAT has two interesting tricks in it. The first trick is the counter test on line 2. The batch file needs to perform ten `IF EXIST COUNT.00 GOTO 00` tests for the values 00-09. Because these tests will be exactly the same except for the digit used, all ten tests are combined into a single FOR-loop. The second trick is the 0-length file.

You can create a 0-length file by typing a file that does not exist and piping the results to a file. Because there's nothing to type, the resulting file is empty. Because the file has a length of zero, it does not take up any disk space, so you get to use the name as a counter holder without any disk space penalty.

You can also use occasional batch files for other kinds of applications. For example, my disk testing program has two levels of testing. The first is the quick mode that takes only a few minutes to run. The complete mode spots more errors but takes much longer to run. The batch file that runs the program normally runs it in quick mode; however, it occasionally runs it in complete mode. You can have as many different occasional batch files as you need. The only trick is to remember to use a different name for each counter.

The approach in OCCASION.BAT works if you only have a fairly short period. However, if you wanted to have a longer period, having a separate section for each possible value of the counter would make for a long and slow batch file. However, it is possible to avoid this, as OCCASIN2.BAT in FIG. 32-7 illustrates.

OCCASIN2.BAT first jumps to other sections if the value of the counter is 0 or if the file is missing. Then it uses two IF-tests to handle the special case of the last counter value where the counter has to be reset to zero. After that, all that is needed is to increment the counter value by 1. It's possible to compute this new counter value by using Batcmd to add one to the environmental variable that stored

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: OCCASIN2.BAT`
`REM PURPOSE: Run Programs Occasionally`
`REM VERSION: 1.00`
`REM DATE: June 5, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`SET MATH=`	If the Math environmental variable already exists, reset its value.
`FOR %%J IN (0 1 2 3 4 5 6 7 8 9) DO IF EXIST COUNT.%%J SET MATH=%%J`	Loop through the acceptable counter values. If the file Count exists with that extension, store the value to the Math environmental variable. To increase the period, just add more items to the loop.
`IF (%MATH%)==(0) GOTO 00`	If the Math environmental variable equals "00" then jump to the section that performs the backup.
`IF (%MATH%)==() GOTO NOFILE`	If the Math environmental variable does not exist, jump to a section to create the counter variable.
`IF (%MATH%)==(9) REN COUNT.9 COUNT.0` `IF (%MATH%)==(9) GOTO END`	If the Math environmental variable equals the maximum value for the counter, reset the counter and exit the batch file. These lines must also be changed to increase the period.
`SET NOW=%MATH%`	Store the value of the Math environmental variable to the Now environmental variable.
`BATCMD AD`	Use Batcmd to increase the value of the Math environmental variable by one.
`REN COUNT.%NOW% COUNT.%MATH%`	Increase the counter by one by renaming the counter file.
`GOTO END`	Exit the batch file.

`:00`
`REM All Work Done Here`
`BACKUP C:\ A:`
`REN COUNT.0 COUNT.1`
`GOTO END`
	When the counter reaches the proper value, perform the backup, reset the counter and exit.

32-7
OCCASION2.BAT also backs up the hard disk once every ten times the computer is rebooted, only it is shorter than OCCASION.BAT in FIG. 33-6.

32-7
Continued.

Batch File Line	Explanation
:NOFILE REM Restore Counter File REM Then Restart Process TYPE NOFILE > COUNT.0 GOTO 00	When the counter file does not exist, create it and jump to the section to perform the backup.
:HELP ECHO OCCASION.BAT Demonstrates How ECHO Commands Can Run Occasionally ECHO By Performing A Backup Every ECHO Tenth Time It Is Run GOTO END	Display a help screen and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.

the counter that matched the extension. Because Batcmd strips off the leading 0 when it performs addition, OCCASIN2.BAT does not use the leading 0 in the counter the way OCCASION.BAT did. For periods longer than five to ten, this approach is much faster.

Using the volume in a batch file

When you format a floppy diskette or hard disk, DOS gives you the option of assigning an eleven-digit volume label to the disk. Occasionally, it can be useful to access this volume label in a batch file. For example, if you sent the user three disks of data to install and a batch file to handle the installation, checking the volume label is one way to make sure the user inserted the correct disk at each stage of the installation. (Checking for a unique filename on each disk is another approach that is much easier.)

DOS does not give you any built-in functions to access the volume label from within a batch file. In fact, all it has is the VOL command to display the volume label on the screen. We can use this as the first of several steps to make the volume available to the batch file. Using the VOL command, we can pipe the information DOS displays into a file with the command

```
VOL > STOREVOL.BAT
```

in a batch file called GETVOL.BAT. The VOL command only displays information and does not need input like the DATE or TIME commands, so we do not need to use piping to send a Return to the command.

If you run VOL from the keyboard, you will see a message like "Volume in drive C is RICHARDSON." When you pipe this to a batch file, the batch file contains the same information.

The trick now is to run STOREVOL.BAT. We can automate this by including a STOREVOL command as the last command in GETVOL.BAT. STOREVOL.BAT immediately issues the command "Volume in drive C is RICHARDSON." This runs VOLUME.BAT and passes it the following information:

```
%1 in
%2 drive
%3 C
%4 is
%5 RICHARDSON
```

So VOLUME.BAT can place the volume into the environment with this command:

```
SET VOLUME=%5
```

To summarize, the steps in the process are as follows:

1. Run GETVOL.BAT. This file first pipes the volume label into the file STOREVOL.BAT and then runs STOREVOL.BAT. Because the CALL command isn't used, control never returns to GETVOL.BAT.
2. STOREVOL.BAT runs and enters the single command VOLUME to run VOLUME.BAT. The volume label is passed as %5. The CALL command isn't used, so control never returns to STOREVOL.BAT.
3. VOLUME.BAT runs and places the contents of the volume label into an environmental variable.

So far, the process has been fairly simple, but you must consider one major complication: Volume labels can have spaces in them. In fact, your volume label could be "a b c d e f," which has five spaces in it. Using the process above, only the "a" part of the volume label would be placed into the environment. Although GETVOL.BAT and STOREVOL.BAT can remain the same, VOLUME.BAT must be modified to handle volume labels with spaces.

Figure 32-8 shows VOLUME.BAT with the necessary modifications. This version issues four SHIFT commands to move the first component of the volume label into the first replaceable parameter position. VOLUME.BAT then loops through all the remaining replaceable parameters, appending a space and the next component onto the VOLUME environmental variable.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: VOLUME.BAT`
`REM PURPOSE: Store Volume In Environment`
`REM VERSION: 2.00`
`REM DATE: December 24, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`SHIFT` `SHIFT` `SHIFT` `SHIFT`	Shift the batch file four times to bring the first component of the volume label into the first replaceable parameter.
`SET VOLUME=%1`	Set the environmental variable Volume equal to the first word in the volume label.
`:TOPLOOP`	Label marking the top of the loop that processes the remainder of the volume label.

32-8
This version of VOLUME.BAT will properly handle spaces in the volume label.

32-8
Continued.

Batch File Line	Explanation
`SHIFT`	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
`IF (%1)==( ) GOTO END`	Once the replaceable parameters are processed, exit the batch file.
`SET VOLUME=%VOLUME% %1`	If the batch file reaches this point, another component of the volume label exists so append it on to the environmental variable that is storing the volume label.

`GOTO TOPLOOP`
`:HELP`
`  ECHO VOLUME.BAT Stores Volume`
`  ECHO In Environment`
`  ECHO Must Be Called By Another`
`  ECHO Program`
`  BATCMD SL`
`  ECHO Can Not Run From Command Line`
`GOTO END`
	Continue looping.
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

When a disk does not have a volume label, DOS displays a "Volume in drive C does not have a label" message. Because nothing in VOLUME.BAT limits the volume label to eleven characters, it ends up creating an environmental variable containing "does not have a label," which is what you might want when the drive does not have a volume.

Of course, once you get the volume label into the environment, you need something to do with it. Typically, it's tested to see if it matches some predefined value. You could do this with a standard IF-test. It's important to remember that many aftermarket utilities allow volume labels with lowercase letters, so you can't make any assumptions about the capitalization. However, if you are testing disks you created, then you know what capitalization to expect.

Because getting the volume label is so difficult in a batch file, Batcmd includes a function to make it easy. Just enter the command

```
BATCMD VO Drive
```

where *Drive* is the drive you want to find the volume label of (drives A–E are the only drives Batcmd will accept), and Batcmd will store its volume label in the environment under the name BATCMD using the same capitalization used for the volume label.

Batch file floppy-disk catalog

Batch files can catalog disks and work fairly well if you have only a few disks. If you have more than a few, however, you're probably better off buying one of the many shareware disk cataloging programs that are available because the batch file method is fairly slow and lacks important error checking. Be careful when you use it and make frequent backups.

Figure 32-9 shows CATALOG.BAT, the batch file that creates the catalog. Notice that it lacks any error-checking at all—I've left this for the user to add. The batch file should determine that it's logged into the proper drive and subdirectory before running. As written, the batch file makes sure the user enters a name and then uses the FOR-loop to cycle through all the files on the A drive. Notice that this would be a good application for grabbing the volume label and including it in the ECHO command.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: CATALOG.BAT`
`REM PURPOSE: Cataloging Floppy Disks`
`REM VERSION: 2.00`
`REM DATE: November 15, 1991`
	Documentation remarks.

`REM Assumes You Are Using The B-drive`
`REM For The Floppies And The C-drive`
`REM For The Catalog.`
	More documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO NONAME`	If the user did not enter a name for this catalog entry, jump to an error-handling routine.
`C:` `CD\CATALOG`	Change to the appropriate drive and subdirectory.
`ECHO Insert Disk To Catalog In A-Drive` `ECHO And Press Any Key When Ready`	Tell the user what to do next.
`PAUSE>NUL`	Pause the program until the user presses a key.

`FOR %%j in (B:\*.*) DO ECHO %%j`
`%1 %2 %3 %4 %5 %6 %7 %8 %9 >>`
`C:CATALOG.TXT`
	For each file on the B-drive, pipe the file name, some spaces and the first nine words of the catalog entry name into the catalog file.

`GOTO END`	Exit the batch file.

`:NONAME`
`ECHO You Must Enter Catalog Name`
`ECHO Where Name Describes The Disk`
`GOTO END`
	When the user does not enter a catalog name, display an error-message and exit the batch file.

`:HELP`
`ECHO Creates A Catalog In The`
`ECHO C:\CATALOG Subdirectory That`
`ECHO Lists All The Files In the`
`ECHO B-Drive Along With The Name`
`ECHO You Enter On The Command Line`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

32-9
CATALOG.BAT creates a catalog of files on a floppy disk.

Figure 32-10 shows REMOVE.BAT, a batch file for removing entries. It also lacks much error checking, but it's impossible to add enough error checking with batch commands. The basic problem is using the FIND command to select lines to delete. If you start the batch file with the replaceable parameter OFFICE, it will not only delete the lines with the added label OFFICE but will also delete the file OFFICE.TAX and any other line containing the line OFFICE. For this reason, this cataloging method must be used with a great deal of care and frequent backups.

32-10
REMOVE.BAT will remove entries from the catalog.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: REMOVE.BAT`
`REM PURPOSE: Removes Entries From Catalog`
`REM VERSION: 2.00`
`REM DATE: November 15, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO NoName`	If the user did not enter a name for this catalog entry, jump to an error-handling routine.
`SET FIND=%1`	Store the first word of the catalog title to search file in the environment.
`SHIFT`	Move all the replaceable parameters down one level.
`:TOPLOOP`	Label marking the top of the loop to process any remaining words in the catalog title.
`IF (%1)==() GOTO CONTINUE`	Once there are no more words in the catalog title to process, jump out of this processing loop.
`SET FIND=%FIND% %1`	When the batch file reaches this point, more words exist for the catalog title so append this word onto the existing environmental variable.
`SHIFT`	Move all the replaceable parameters down one level.
`GOTO TOPLOOP`	Continue looping.
`:CONTINUE`	Label used to jump out of the above loop.

Batch File Line	Explanation

`ECHO All Floppy Catalog Entries Containing`
`ECHO %FIND% Will Be Deleted`
`ECHO If This Is Not OK, Press Control-Break`
`ECHO Otherwise, Press Any Other Key`
`PAUSE`
	Tell the user what will happen next.

`C:` `CD\CATALOG`	Change to the appropriate drive and subdirectory.
`COPY CATALOG.TXT CATALOG.BAK`	Make a backup copy of the catalog before modifying it.
`TYPE CATALOG.TXT|FIND/V "%FIND%" > JUNK.TMP`	Type the catalog, use the Find filter to search for all lines not containing the catalog text to delete and pipe the results to an output file.
`DEL CATALOG.TXT`	Delete the current catalog.
`REN JUNK.TMP CATALOG.TXT`	Rename the temporary file to make it the catalog file.
`GOTO END`	Exit the batch file.

`:HELP`
`ECHO This Batch File Deletes The`
`ECHO Entries From The Disk Catalog In`
`ECHO The C:\CATALOG Subdirectory That`
`ECHO Contain The Name You Enter On The`
`ECHO Command Line`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

32-10
Continued.

Adding to & deleting from your path

Most people simply include every subdirectory they need DOS to search through in the path and forget it. There are occasions, however, when you might want to have two separate paths (for example, if you're testing a new program and you don't want DOS to search anywhere other than the current subdirectory; if you have added a subdirectory for a special program and you want it in the path for only a short period of time; or if you have so many subdirectories to search that you want to have a separate path for each task). The plan of attack is the same for each of these problems. The steps are as follows:

1. Store the current path to a variable in the environment.
2. Replace the path with a new path or modify the existing path.
3. Restore the old path from the environment.

A batch file can store the current path to another environmental variable with the command:

```
SET OLDPATH=%PATH%
```

Of course, at this point, two versions of your path exist—one stored under OLDPATH and one stored under PATH—so you will need an expanded environment if you have a long path.

If you want to create a new path, it will be easier to either write a custom batch file for each alternative path or simply enter it from the command line. However, if you have two or three alternative paths that you use a lot, you might want to have a batch file that uses a replaceable parameter to let you select between them. For example,

```
@ECHO OFF
IF (%1)==(1) SET PATH=C:\;C:\DBASE
IF (%1)==(2) SET PATH=C:\;C:\WORD
IF (%1)==(3) SET PATH=C:\;C:\NORTON
```

would let you switch rapidly between three different paths. You could even use this same batch file to restore the original path with two modifications:

```
@ECHO OFF
IF (%1)==()  SET PATH=%OLDPATH%
IF (%1)==()  SET OLDPATH=
IF (%1)==(1) SET PATH=C:\;C:\DBASE
IF (%1)==(2) SET PATH=C:\;C:\WORD
IF (%1)==(3) SET PATH=C:\;C:\NORTON
```

A batch file can also remove subdirectories from your path, although the process is a little tricky. A batch file has access to the %PATH% environmental variable but has no way to strip it into its component parts. However, the different subdirectories in a path are separated by semicolons, which DOS treats as legal dividers for replaceable parameters. If you could issue the command

```
BATCH %PATH%
```

and have DOS treat %PATH% as the environmental variable, then the first subdirectory in the path would be %1, the second would be %2, and so on. Unfortunately, you can't use environmental variables on the command line in this fashion. Luckily, batch files can, so if one batch file invokes another with this exact same command, the subdirectory components are passed as separate replaceable parameters—just as you would expect.

Putting all this together, we can come up with a very nice set of batch files for editing the path. The first is called EDITPATH.BAT. While it contains command line help and internal documentation, its single operation line is

```
EDIT2 %PATH%
```

which calls EDIT2.BAT and passes it the path on the command line. This causes EDIT2.BAT, shown in FIG. 32-11, to be passed the entire path as a series of replaceable parameters.

EDIT2.BAT loops through the subdirectories one at a time and asks the users if they want to keep that subdirectory in the path. If they answer yes, it stores that subdirectory in an environmental variable. If they answer no, it does not store that subdirectory. When all of the subdirectories have been processed, EDIT2.BAT constructs the new path.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: EDIT2.BAT`
`REM PURPOSE: Selectively Delete`
`REM Path Subdirectories`
`REM VERSION: 3.00`
`REM DATE: December 14, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`IF (%1)==() GOTO HELP`	Since EDIT2.BAT is not designed to run from the command line it should never run with a value for "%1" so when that happens jump to a help section.

`REM The Environmental Variable`
`REM P Will Contain The New Path`
`REM We Must Maintain Old Path`
`REM So Utilities Will Work`
`REM P Is Used So "SET P=" Is As`
`REM Close In Length To "PATH="`
`REM As Is Possible`
	Documentation remarks.

`SET P=`	Make sure the environmental variable is empty.

`REM MATH Environmental Variable`
`REM Used To Track Additions Since`
`REM First Addition To Path Must`
`REM Skip Leading Semi-Colon`
	Documentation remarks.

`SET MATH=0`	Set the flag environmental variable to zero.
`:TOP`	Label marking top of loop.
`IF (%1)==() GOTO ENDLOOP`	Once there are no more replaceable parameters, jump out of the loop.
`BATCMD SL`	Skip a line on the screen using Batcmd.
`BATCMD YN Keep %1 in PATH (Y/N)`	Ask a question using Batcmd.
`IF NOT ERRORLEVEL 1 GOTO SKIP`	If the user answered "no" then skip keeping the subdirectory.
`BATCMD AD`	Increase the flag value by one.
`IF %MATH%==1 SET P=%1`	If the flag has a value of one (0+1) then this is the first directory in the path so do not precede it with a semicolon.
`IF %MATH%==1 GOTO SKIP`	If the flag has a value of one, then nothing else needs to be done so jump to end of loop.

32-11
EDIT2.BAT lets you decide whether or not to keep each subdirectory in your path.

32-11
Continued.

Batch File Line	Explanation
`SET P=%P%;%1`	The path already has at least one directory so append this one and precede it with a semicolon.
`:SKIP`	Label used to bypass part of the code in the loop.
`SHIFT`	Move all the replaceable parameters down one level.
`GOTO TOP`	Continue looping.
`:ENDLOOP`	Label marking the end of the loop.
`PATH %P%`	Set the path equal to the contents of the "P" environmental variable.
`ECHO PATH Now Set To:` `PATH`	Tell the user the new path. Since the path could be very long, it must be displayed separately because including it in the ECHO command might make the command too long.
`SET MATH=` `SET P=`	Reset the environmental variables used by the batch file.
`GOTO END`	Jump to the end of the batch file.

`:HELP`
`ECHO This Batch File Is Used By`
`ECHO EDITPAT2.BAT To Edit The Path`
`ECHO *DON'T RUN FROM COMMAND LINE**`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

As you elect to keep each subdirectory, EDIT2.BAT stores that subdirectory in a temporary environmental variable called "P." Only when you finished making all your selections is the contents of the PATH environmental variable replaced with the contents of the P environmental variable.

There are two problems that EDIT2.BAT has to avoid. First, the first subdirectory added to the path is not preceded by a semicolon, while all the remaining subdirectories are. Second, the length of any command line cannot exceed 127 characters, yet the path itself can be 122 characters. (Five characters are needed for the "PATH " that precedes the path when you create it.

When I first started writing this batch file, I tried to avoid adding the semicolon before the first subdirectory with these two IF-tests:

```
IF NOT (%PATH%)==() SET PATH=%PATH%;%1
IF (%PATH%)==() SET PATH=%1
```

This approach works well as long as the path is fairly short. However, the path environmental variable is used twice in the first IF-test so a moderately long path

causes this statement to exceed the 127-character limit DOS places on the command line. EDIT2.BAT avoids this problem by setting the MATH environmental variable to 0 and then incrementing it with Batcmd for every subdirectory added. The leading semicolon is avoided only when MATH has a value of 1.

EDIT2.BAT constructs a second environmental variable called P to contain the modified path until the end of the batch file when the path is replaced with the contents of the P environmental variable. The reason for this is quite pragmatic. If the path was reset at the beginning of the batch file and reconstructed in the batch file—rather than using P—then Batcmd would not run because it is no longer in your path!

Password protection

One way to keep casual users from accessing your hard disk is to add password protection. Password protections range from simple programs that require you to enter a password to complex systems that encrypt the file allocation table (FAT) and won't access the hard disk at all until the proper password is entered. If all you need is light protection, then you can put together a batch file password system that's very effective.

The password application we'll be looking at is simply a batch file that requires a password. Once the user enters a password, it terminates. You could add the commands to the batch file to run a specific application or you could add this code to the top of your AUTOEXEC.BAT file. Keep in mind that this system won't keep a user from booting off a floppy disk and then accessing your hard disk. In addition, because you can't encrypt the batch file, anyone with access to the hard disk can look at the batch file and figure out your password. They will have to have some experience to do this, though, because the password is stored using its ASCII value. With that caveat in mind, let's construct the password system.

The password batch file, PASSWORD.BAT, is shown in FIG. 32-12. It uses Batcmd to get the first character of the password. If that character is correct, it uses Batcmd to get the second character. If the first character is incorrect, it increments a counter and asks again. It repeats this process for the second and third character. Any time an incorrect character is entered, the program begins prompting for the password from the beginning. Once the counter reaches three—which is after you've entered three incorrect passwords—the batch file will enter an endless loop and appear to lock up.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: PASSWORD.BAT`
`REM PURPOSE: Batch Password`
`REM VERSION: 1.51`
`REM DATE: March 25, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`CLS`	Clear the screen.
`CTTY NUL`	Turn off the console. While this is in effect, the keyboard does not work and nothing shows up on the screen without piping.

32-12
PASSWORD.BAT requires the user to enter the correct password (which is RON at the moment, as currently set up) in three tries or less.

32-12
Continued.

Batch File Line	Explanation
`SET COUNTER=1`	Create a counter variable that will be used to count the number of password attempts.
`ECHO Enter Password > CON`	Tell the user to enter the password. This shows up on the screen only because it is piped there.

`REM Enter Three Digit Password Here`
`REM A=ASCII Value Of First Digit`
`REM C=ASCII Value Of Second Digit`
`REM E=ASCII Value Of Third Digit`
`REM Run FINDPASS.BAT To Find These`
`REM If You Are Not Sure What To Use`
	Documentation remarks.

`REM ------------------------------`
`REM Change These Numbers`
`REM To Change Password`
`REM ------------------------------`
	Documentation remarks.

`SET A=82`
`SET C=79`
`SET E=78`
	Create the three variables that contain the ASCII values of the three digits of the password.

`REM ------------------------------`
`REM Need To Develop The Second Number`
`REM To Use In The Dual If-Tests`
	Documentation remarks.

`SET MATH=%A%`
`BATCMD AD`
`SET B=%MATH%`
`SET MATH=%C%`
`BATCMD AD`
`SET D=%MATH%`
`SET MATH=%E%`
`BATCMD AD`
`SET F=%MATH%`
`SET MATH=`
	The dual If-tests need both the ASCII value of the password digit and the value one higher--for the "IF NOT ERRORLEVEL" portion of the test. Rather than having the user enter a value one higher, this section uses the addition abilities of Batcmd to calculate those values.

`:TOP`	Label marking the top of the section that gets the first digit of the password.
`BATCMD GK<CON`	Use the Batcmd to obtain any character from the user. This only works because the input is piped from the keyboard.
`IF ERRORLEVEL %A% IF NOT` `ERRORLEVEL %B% GOTO SECOND`	If the user entered this digit properly, jump to the section to get the second digit.

`IF %COUNTER%==1 GOTO 1`
`IF %COUNTER%==2 GOTO 2`
`IF %COUNTER%==3 GOTO 3`
	Jump to a section to increment the counter. This section also automatically returns the user to the section to enter the first digit of the password.

Batch File Line	Explanation
`GOTO TOP`	The batch file never reaches this point, this line is just used to having a "closing" point for the loop.
`:SECOND`	Label marking the top of the section that gets the second digit of the password.
`ECHO X>CON`	Display an X on the screen to represent the first digit of the password.
`BATCMD GK<CON`	Use the Batcmd to obtain input from the user.
`IF ERRORLEVEL %C% IF NOT ERRORLEVEL %D% GOTO THIRD`	If the user entered this digit properly, jump to the section to get the third digit.

`IF %COUNTER%==1 GOTO 1`
`IF %COUNTER%==2 GOTO 2`
`IF %COUNTER%==3 GOTO 3`
	Jump to a section to increment the counter.

`GOTO SECOND`	The batch file never reaches this point, this line is just used to having a "closing" point for the loop.
`:THIRD`	Label marking the top of the section that gets the third digit of the password.
`ECHO XX>CON`	Display an XX on the screen to represent the second digit of the password.
`BATCMD GK<CON`	Use the Batcmd to obtain input from the user.
`IF ERRORLEVEL %E% IF NOT ERRORLEVEL %F% GOTO CORRECT`	If the user entered this digit properly, jump to the section to leave the batch file.

`IF %COUNTER%==1 GOTO 1`
`IF %COUNTER%==2 GOTO 2`
`IF %COUNTER%==3 GOTO 3`
	Jump to a section to increment the counter.

`GOTO THIRD`	The batch file never reaches this point, this line is just used to having a "closing" point for the loop.

`:1`
`ECHO Wrong Password, Start Over>CON`
`SET COUNTER=2`
`GOTO TOP`
	Section to handle the first time the user enters the password wrong. Increments the counter by one, tells the user about the mistake and jumps to the top of the password routine.

32-12
Continued.

32-12
Continued.

Batch File Line	Explanation

`:2`
`  ECHO Wrong Password, Start Over>CON`
`  SET COUNTER=3`
`GOTO TOP`
	Section to handle the second time the user enters the password wrong. Increments the counter by one, tells the user about the mistake and jumps to the top of the password routine.

`:3`	Label marking the top of the section used to lock up the computer for an invalid password.
`CLS > CON`	Clear the screen. The piping makes this work.
`BATCMD PC 12 38 > CON`	Position the cursor with Batcmd. Again, the piping makes this work.
`ECHO --DEATH-- > CON`	Pipe a message to the screen.
`:DEATH` `GOTO DEATH`	This endless loop locks up the computer. Since the keyboard is not working, the user cannot break out.
`:CORRECT`	Label marking the beginning of the section where the user entered the password properly.

`  FOR %%J IN (A B C D E F COUNTER)`
`  DO SET %%J=`
`  SET COUNTER=`
	Reset the environmental variables used by the batch file.

`  CTTY CON` `GOTO END`	Turn the console back on and exit the batch file.

`:HELP`
`  ECHO This Demonstration Batch File`
`  ECHO Requires The User To Enter A`
`  ECHO Password Before It Continues`
`  ECHO And The User Can Not Break Out`
`  ECHO By Pressing Control-Break Or`
`  ECHO Any Other Method`
`  BATCMD SL`
`  ECHO The Demonstration Password Is`
`  ECHO RON And The Capitalization`
`  ECHO Must Match Exactly`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

With any batch file, you have the problem of the user pressing Ctrl-Break to stop the batch file. PASSWORD.BAT avoids this problem almost completely by using the CTTY NUL command at the top. This turns off the console so that the computer won't accept most inputs from the keyboard and won't write output to the screen. It's interesting to note that the batch file will still accept the Ctrl-Break, but won't accept the "y" in response to the Halt Batch Process (y/n) message—effectively

locking the computer. Notice that, after the correct response is entered, the batch file restores the screen and keyboard with the CTTY CON command.

Because the keyboard and screen don't respond to the batch file under the CTTY NUL command, the batch file forces them to work by piping output to the screen with the >CON piping command and grabbing input from the keyboard with the <CON piping command.

If you add the password to the top of your AUTOEXEC.BAT file, anyone booting the computer from the hard disk will have to enter a password. It should be at the very top of the AUTOEXEC.BAT file so that the user won't have time to enter Ctrl-Break before the password part of the AUTOEXEC.BAT file takes over. You should also create a stand-alone batch file with a name like LOCK.BAT so you can lock your computer but leave it running when you have to leave it.

Choosing password batch file codes

One of the problems of using PASSWORD.BAT is you have to know the ASCII values of the three digits of the password in order to configure the batch file. You could use an ASCII chart, but a batch file can automate the process. FINDPASS.BAT in FIG. 32-13 will prompt you for the three characters you want to use as a password and tell you the values to enter into PASSWORD.BAT to configure it for the desired password.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: FINDPASS.BAT`
`REM PURPOSE: Find Password Errorlevel`
`REM PURPOSE: Values For PASSWORD.BAT`
`REM VERSION: 2.00`
`REM DATE: June 6, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`ECHO Enter First Character Of Password`	Tell the user what to do.
`BATCMD GK`	Use Batcmd to get the keystroke and set error-level accordingly.
`CALL SAYERROR`	Call SAYERROR.BAT to store the errorlevel in the environment.
`SET A=%ERROR%`	Store the errorlevel in another environmental variable.
`ECHO Enter Second Character Of Password`	Tell the user what to do.
`BATCMD GK`	Use Batcmd to get the keystroke and set error-level accordingly.
`CALL SAYERROR`	Call SAYERROR.BAT to store the errorlevel in the environment.
`SET C=%ERROR%`	Store the errorlevel in another environmental variable.

32-13
FINDPASS.BAT prompts you for the password you want to use in PASSWORD.BAT and then tells you the setting to use for the three environmental variables A, C, and E used in PASSWORD.BAT to store the three digits of the password.

32-13
Continued.

Batch File Line	Explanation
`SET C=%ERROR%`	Store the errorlevel in another environmental variable.
`ECHO Enter Third Character Of Password`	Tell the user what to do.
`BATCMD GK`	Use Batcmd to get the keystroke and set error-level accordingly.
`CALL SAYERROR`	Call SAYERROR.BAT to store the errorlevel in the environment.
`SET E=%ERROR%`	Store the errorlevel in another environmental variable.

`ECHO In PASSWORD.BAT Set A=%A%`
`ECHO Set C=%C%`
`ECHO Set E=%E%`
`GOTO END`
	Tell the user what values to use in PASSWORD.BAT and exit the batch file.

`:HELP`
`ECHO Tells You The Values For`
`ECHO PASSWORD.BAT`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

`SET A=`
`SET C=`
`SET E=`
`SET ERROR=`
	Before leaving the batch file, delete the environmental variable it used.

Summary

EDITBAT.BAT lets you keep all your batch files in the C:\BAT subdirectory yet still easily edit them on-the-fly.

NOBURN.BAT is a simple screen saver that runs from DOS and requires so little memory you can run it while shelled out of another program.

CHECKWIN.BAT tells you if Windows is running without the need for a utility program.

One way to change subdirectories quickly is with a special utility like NCD in the Norton Utilities or QC from Steven Flores.

Another way to change to a single subdirectory quickly is with a custom batch file.

A batch file like RETURN.BAT can store the current subdirectory in a batch file for instant return.

A batch file like RETURN2.BAT can store a single subdirectory in the environment for quick changes.

A batch file like RCD.BAT can switch between a number of subdirectories quickly.

Programs that inflexibly store their configuration in a single file can be run in different configurations by making multiple copies of that configuration file and using batch files to activate different copies of the configuration file.

Combining the ATTRIB command with a batch file gives you a quick and powerful file location program.

By having the AUTOEXEC.BAT "count" reboots by using the name of a 0-length file, you can run programs occasionally by running them only after a certain number of reboots.

A batch file like GETVOL.BAT can pipe the volume label into the environment, where you can perform If-tests on it to verify that the proper floppy disk has been inserted in the drive.

Passing the path between batch files as a replaceable parameter gives the second batch file access to the subdirectories in the path as individual replaceable parameters.

A batch file like EDITPATH.BAT can let you pick subdirectories individually to keep in the path or discard from the path.

A batch file like PASSWORD.BAT can set up a surprisingly powerful password protection program using a batch file.

33 Applications and problems

You have finished most of the book. All that remains are the sections on advanced topics, configuration and the reference section. Because we have covered a lot of material, now is a good time to stop and review what we've learned. What follows are four quick problems; take a few moments and try to work them. It's important that you have the skills necessary to solve these problems before you go on. If you have difficulty with these problems, you might want to look back through the book before going on to the section on configuration.

Problem 10

A friend asks you if it's possible to put together a telephone database using batch files. She is looking for a system for a system where she can run the batch file followed by a "nickname" of the person to call and have her modem dial the number for her. As she sees it, the command `PHONE MOM` should call her mother.

You decide to put together a "mock-up" first with just a few numbers like information and 911 first to make sure she likes the system. As you begin your reseach, you find out two pieces of critical information:

- A command like `ECHO ATDT 911 > COM1` will dial the phone. The ATDT tells the modem to dial, the 911 is the phone number, and the >COM1 pipes everything to the port to which the modem is connected. Note that this might not work on some high-speed modems.
- Once the modem has finished dialing and the user has picked up the receiver, the command `ECHO ATH > COM1` will cause the modem to remove itself from the circuit without hanging up the connection.

Problem 11

A friend of yours recently bought his son a computer for his high school graduation. His son will use the computer in college to write term papers and do his homework.

Your friend does not expect his son to create a lot of critical data, so he is not worried about performing backups too often. Just to be sure, though, he has asked you to write a batch file that will list the files that need to be backed up so that he can decide if it's time to perform a backup.

Problem 12

Your boss recently purchased a laptop computer. He likes the computer and keyboard but has trouble changing subdirectories because the backslash key is located in an awkward position. You have decided to try to win favor with the boss by writing a batch file to make it easier for him to change subdirectories without the need to use the backslash key.

You are looking for a system that will work with any subdirectory, mainly because you do not know how your boss has his hard disk structured. That lets out a system where your boss runs the batch file with a nickname for the most common subdirectories.

Problem 13

You have started trying to learn to use DOS piping. As you experiment, you find a lot of 0-length files appearing on your hard disk, so you have decided to write a batch file to automatically delete all the 0-length files in the current subdirectory.

Answers

You have a number of ways to tackle each of these problems. My solutions are shown in Appendix A in the back of the book. It is not important that you match my answers; if you've developed a working solution to the problem, your solution is just as good as mine and maybe even better.

Part seven

Advanced batch file topics

By now, you can call yourself a batch file master. Anyone making it this far in this book has a clear idea of what batch files can do and can write some very powerful batch files. These two chapters are not necessary to writing useful batch files but they are necessary if you want to write the most powerful batch files possible.

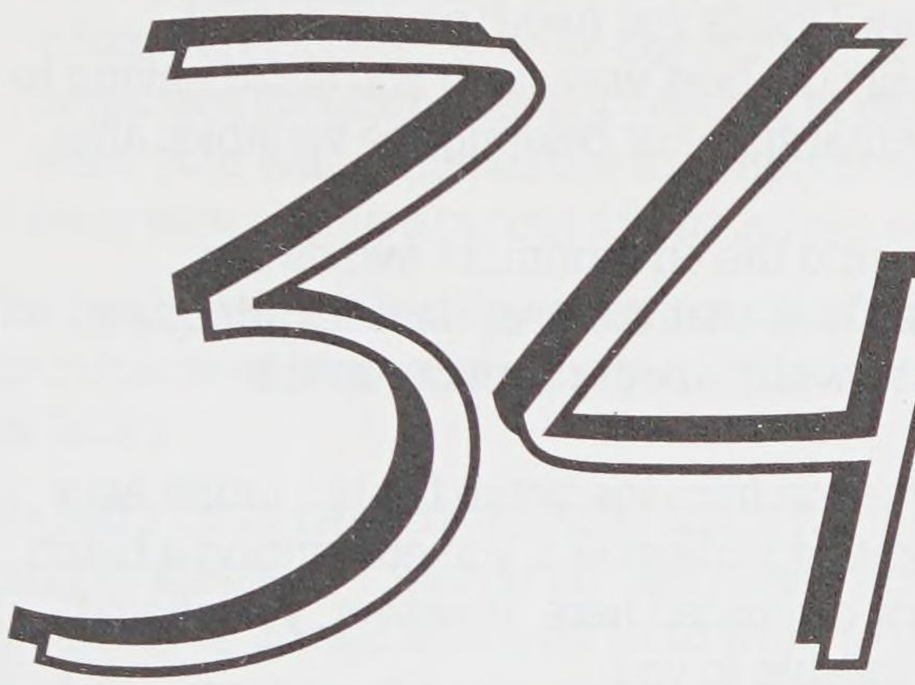
Writing your own batch file utilities Writing your own batch file utilities is not as hard as you think. In fact, it is often one of the easier programming tasks you can tackle. This chapter shows you how to write several simple batch file utilities in QBasic, Builder Lite and Pascal. Even if you do not "speak" either of these three languages, the insights this chapter offers for programming batch file utilities can make that task easier in any language.

Chapter 34

Modifying DOS with DOSKEY & batch files There is a lot of interest in changing the names of DOS commands or even changing the way they work. Of course, with external commands, you just rename the program file to change the name of the command (e.g., rename FORMAT.COM to PREPARE.COM and instantly the command to format a disk on your system is "Prepare"), but that does not work for internal commands and does not change the way the commands work.

Chapter 35

In earlier versions of this book, I had a chapter on going into COMMAND.COM with DEBUG and physically altering the names of internal commands. I've dropped that with this book because DOSKEY makes all that unnecessary. Because DOSKEY macros are run before internal commands, a DOSKEY macro named "DEL" takes priority over the internal command. If that DOSKEY macro remaps the command to a batch file, then the batch file alters the way the command works. Plus, you can map other command names, like "KILL", to the same batch file to alter command names. All in all, DOSKEY macros are a very neat way to alter how DOS functions.



Writing your own batch file utilities

One of the things you have seen quite clearly in this book is that batch files need help. While the batch language is useful, there are many necessary things like counting that it simply does not do. When your needs exceed the ability of the batch language, then a possible solution is to write your own batch file utility.

Wait, don't turn the page!

While it sounds like you need to be an experienced programmer with lots of experience, batch file utilities are some of the easiest programs you can write. They are so easy because most batch file utilities perform just one thing. Look at the CHOICE.COM that Microsoft added to DOS 6. All it does is display a prompt and get a single character from the user and place it in the errorlevel. Many batch file utilities are simpler than that.

What is a batch file utility?

What is a batch file utility? One answer to this question is any program that runs within a batch file. Yet, that is not a very good answer because it would make your word processor, database, and spreadsheet batch file utilities! My answer is this: a batch file utility is any program designed to take information from a batch file or provide information to the batch file in order to enhance its operation. In order to do this, a batch file utility must be able to interface with the batch file.

As part of this chapter, we will see how to interface three languages—QuickBASIC, Builder and Pascal—with the batch language. We will also develop two useful batch utilities in each language. As you will see, the process is fairly painless.

What it needs

In order to write a batch utility, you need two things, a way to get information from the batch file to the utility and a way to get information from the utility to the batch

file. In between getting the information into the batch utility and getting it out, the program works exactly like any other program you might write in that language.

There are four ways to get information into a batch file utility:

1. The command line. A DOS command line is limited to 127 characters and must contain the name of the utility program to run so this restricts the amount of information that can be passed this way. Unless you need to pass a lot of information to the utility, this is generally the best way since you avoid having to worry about having enough environmental space or deleting the variables after you use them.
2. The environment. Placing information into the environment avoids the command line length limitation but requires that the user have an adequate environment. It also means you have to worry about cleaning up the environment after you are done.
3. Ask the user. This does not take information from the batch file but rather asks the user a question. This works the same regardless of if you are writing a batch file utility or general program so it is not discussed here. However, you should remember that this option is always available to you.
4. Poll the equipment. Like asking the user, this does not take information from the batch file and works the same in a standard program or batch file utility. For that reason, it is not covered here.

Of course, not all batch file utilities will require input information. For example, a password protection program might have the password compiled into the program itself so all that is required to use it is to run it. A menu program might have the menu options hardwired into the code, so it too would require no inputs.

There are four ways to get information out of a batch file utility and back into the batch file:

1. Store a number in errorlevel. The errorlevel is one byte of information that can contain a number 0–255. It was specifically designed for utilities to pass information back to a batch file. All batch files and programs access the same errorlevel. The password protection program might pass back a 1 if the password is correct and a 0 if it was wrong, or the menu program might pass back the option selected as an errorlevel.
2. Store the results in the master copy of the environment. Because the errorlevel only has room for one byte of information, it is not useful for utilities that need to pass back a lot of information. For example, a utility to multiply two numbers together could not use the errorlevel unless its results were restricted to integers 0–255. One way to avoid this is to pass back the information in the environment. However, batch file utilities get a copy of the environment, and that copy is erased when they terminate, so they cannot use their copy of the environment for this. As we will see, some languages have the ability to search through memory and find the master copy of the environment and make their changes there, thus making sure the environmental variables they create remain available after they terminate.
3. Write an on-the-fly batch file to store the results in the master copy of the environment. Not all languages have the ability to find the master copy of the environment in memory and make their changes there. However, almost every language—including the batch language—has the ability to create disk files.

These languages can store information in the environment by creating a temporary batch file with a single line like `SET ANSWER=2234`. The batch file that runs the utility can CALL this temporary batch file after running the utility to place the results into the environment. As we will see, QuickBASIC must do this to pass back information that will not fit in errorlevel.

4. Print the results to the screen or printer. Because this does not make the information available to the batch file and works the same for a batch file utility or general program, it is not covered in detail here. Nevertheless, you might often want your batch file utilities to simply perform fancy screen formatting or otherwise produce output for the screen so keep this option in mind.

The method you select for getting output from your batch file utility will depend on the purpose of your utility and the limitations of the language you select to write the utility.

QuickBASIC

We will look at two QuickBASIC batch file utilities because Basic is the closest thing available to a "universal" language. However, the QuickBASIC compiler lacks the ability to access errorlevel or to write to the master copy of the environment, so all of its output must be passed back to the batch file using a temporary batch file.

The first batch file utility will take the number passed to it on the command line, add one to it and pass the results back to the batch file. The second utility will take two numbers passed to it via the environment and multiply them together. The numbers can be whole or decimal numbers. Even if QuickBASIC could access the errorlevel, this value could not be passed back via errorlevel.

Adding 1 to command line number

The command to read in the command line in QuickBASIC is

```
Line$ = COMMAND$
```

This command only works in the QuickBASIC compiler and not the QBasic interpretative version that comes with DOS. The command line is read in as a single line without the name of the program, so the command

```
UTILITY/? 123
```

would cause Line$ to have a value of `/? 123`. Note that Line$ is a variable name and can be any legal string name you choose.

Once we have the command line, we need to check to make sure the user actually entered something. The code to do that is shown next:

```
Length = LEN(line$)
IF Length = 0 THEN
  PRINT "Invalid Syntax, Enter 'ADD /?' For Help"
  BEEP
  END
END IF
```

Because the COMMAND$ function does not include the name of the program in the command line, the length of the command line will be 0 if the user did not enter something on the command line. That represents an error because the utility expects a number to add 1 to, so it displays an error message and exits.

Next, we have to loop through the command line looking for just the first piece of information. If the user starts our ADD.EXE utility with the command:

```
ADD 123 UP 456
```

then only the first "replaceable parameter" has any meaning to us. To do this, we compute the length of the Line$ string and then loop through it one character at a time. Once we find a space, comma, tab or semicolon, then we have found the end of the first input. At that point, we can jump out of the loop. The length was computed earlier; the code to do the rest of this is

```
FOR I = 1 TO Length
 Char$ = MID$(Line$, I, 1)
 IF (Char$<>" " AND Char$<>CHR$(9) AND Char$<>",") THEN
 IF First = 0 THEN Build$ = Build$ + Char$
 ELSE
 First = 1
 END IF
NEXT I
```

The variable First starts off with a value of 0. It functions as a flag. Its value is 0 while the first input is being processed, so each character is appended onto the end of a holding variable, Build$. As soon as the first space, tab, or comma is found, the value of First is changed to a 1 and the remaining characters are ignored. That way, Build$ contains only the first input.

As we will see with the next utility, you could just swap and begin appending characters onto another variable if you needed to pass the utility more than one input. You might use this option if you wanted the utility to add two numbers together or if you wanted the user to be able to enter the name of the output environmental variable rather than the utility just using "Answer."

Much of the rest of the code involves straight QuickBASIC programming to perform error checking and perform the addition. This is not a QuickBASIC programming book, so I don't plan on covering it in detail; but we'll go through it quickly.

Now that we have the first input, we can check to see if the user is requesting help with the command

```
IF Build$ = "/?" OR Build$ = "?" THEN
```

If so, we will print the help messages and then exit the utility without performing any addition with the commands:

```
IF Build$ = "/?" OR Build$ = "?" THEN
PRINT "Adds One To Number Passed ";
PRINT "To It On The Command Line"
PRINT "And Creates JUNK.BAT With ";
PRINT "The Command:"
PRINT
PRINT "SET ANSWER=#"
PRINT
PRINT "Where # Represents The Answer. ";
PRINT "That Way, A Batch File Can ";
```

```
    PRINT "Easily Get The Answer Into ";
    PRINT "The Environment"
    END
  END IF
```

Notice that the END command on the next to last line causes ADD.EXE to immediately terminate and return control to DOS or to the batch file that called it.

Now that we know the user did not request help, we can convert the string variable Build$ that contains the input value, add 1 to it, and move on to the output stage. You could do this in just a couple of lines, but I don't recommend it. The utility needs several layers of error checking to make sure the user is not trying to misuse the utility.

The first step is to convert the string to a number with the command `Number = VAL(Build$)`. If the user entered a number, this conversion went just as we would expect. If the user entered anything other than numbers, say `ADD One`, then Number has a value of 0. If the user entered a mixture of numbers and letters, say `ADD 123ABC`, then Number has a value equal to all the numbers up to the first letter—or "123" in this example. We will perform error checking to prevent non-numeric data like "ONE" but not for mixed data like "123ABC" because this is just an example. Users constructing utilities for use by inexperienced users should consider all possible inputs when validating values.

The code to perform the error checking and the addition follows:

```
  IF Number = 0 THEN
    IF ASC(Build$) = 48 THEN
        Number = 1
    ELSE
          PRINT "Invalid Input"
      PRINT "Enter ADD /? For Help"
          BEEP
      END
    END IF
  ELSE
    Number = Number + 1
  END IF
```

If Number has a value of 0, then it checks to see if the first character of the string variable is a "0" with the command "IF ASC(Build$) = 48" and if so, stores one to Number since zero is a valid input and 0 plus 1 is 1. If Number was 0 and the first character was not a 0, then the user entered a non-numeric character as the first character. The utility prints out an error message, beeps the speaker and exits without creating the output batch file. If Number was not 0, then it adds 1 to its value. Note that this addition does not require a whole number, so ADD.EXE will work with non-integer numbers.

We are almost ready to create our output file, but a quirk in QuickBASIC causes us to have to perform some extra work first. Anytime you need to output a numeric answer, you will have the same problem with QuickBASIC. QuickBASIC adds a leading space when printing non-negative numbers, so 47 is printed as " 47"; thus, the command `SET ANSWER=47` becomes `SET ANSWER= 47`, which is clearly not

what we want. Part of the solution is to convert Number to a string, but even that does not clear up the problem entirely because QuickBASIC pads this string with leading spaces. The remainder of the solution is to strip off the leading spaces. The code to do all this follows:

```
Answer$ = STR$(Number)
Answer$ = LTRIM$(Answer$)
```

We are now ready to create our output file. I've used the name JUNK.BAT for the file and the utility does not perform any error checking to keep from overwriting JUNK.BAT. (Note: the commands I'm using to create JUNK.BAT will overwrite any existing file rather than appending to it.) This error checking could be performed in either the QuickBASIC program or the batch file. I've skipped this since on my system I assume that files named JUNK*.* are subject to be overwritten at will and never use them for anything permanent. Again, when using this approach to write utilities for inexperienced users you would be well advised to check to make sure the file exists and ask the user about overwriting it. The code to create the file follows:

```
OPEN "JUNK.BAT" FOR OUTPUT AS #1
 PRINT #1, "SET ANSWER="; Answer$
CLOSE
```

As you can see, the process of creating ADD.EXE is mostly QuickBASIC programming with only a little extra thrown in to facilitate communication with the batch file. To illustrate ADD.EXE in action, SHOWADD.BAT uses it to loop through twenty loops. The operational code for SHOWADD.BAT follows:

```
@ECHO OFF
ECHO Will Loop Through Loop 20 Times
SET Answer=0
:TOP
ADD %Answer%
CALL JUNK.BAT
DEL Junk.BAT
ECHO Loop Number %ANSWER%
IF NOT %ANSWER%==20 GOTO TOP
ECHO Loops Done
SET ANSWER=
```

The files ADD.BAS, ADD.EXE and SHOWADD.BAT are all on the disk in the subdirectory \BAT\QB-ADD. That version of SHOWADD.BAT has documentation and offers command line help.

Multiplying two numbers together

This utility will take two numbers from the environment and multiply them together. We could, of course, take these two numbers from the command line because it is unlikely that two numbers would cause the command line to exceed 127 characters; this approach is more for illustration than a practical application.

Even though MULTIPLY.EXE takes its usual inputs from the environment, it has to start out reading the command line. It does this to check to see if the user used "/?" or "?" to request help. If so, it displays a help message and exits. It must do this first because the routines that follow display error messages if the required environmental variables do not exist, and you would not want that to happen for a

user requesting help. Other than the help message, this code is identical to the code in ADD.BAS, so it is not shown here.

Next, it reads the two environmental variables NUMBER1 and NUMBER2 with the commands:

```
Input1$ = ENVIRON$("NUMBER1")
Input2$ = ENVIRON$("NUMBER2")
```

If either of the environmental variables is missing, these commands do not cause a problem; but the associated string variable is not created, so if NUMBER1 does not exist, then Input1$ is not created. This is a violation of the way the utility is supposed to work and will create problems later on, so the utility must check for this condition. The code to check Input1$ follows:

```
IF Input1$ = "" THEN
 PRINT "NUMBER1 Environmental Variable Missing"
 BEEP
 END
END IF
```

The code for Input2$ is identical except that, of course, Input1$ is changed to Input2$ and the error message specifies "NUMBER2."

Like ADD.BAS, MULTIPLY.BAS must check to make sure that numbers and not letters were entered. The code is very close to the code in ADD.BAS and is shown below for Input1$:

```
IF Number1 = 0 THEN
 IF ASC(Input1$) <> 48 THEN
      PRINT "Invalid Input For NUMBER1 ";
      PRINT "Environmental Variable"
      PRINT "Enter MULTIPLY /? For Help"
      BEEP
      END
 END IF
END IF
```

Number2 is handled in a similar fashion.

Next, the two numbers are multiplied together and converted to a space-free string with the commands

```
Answer$ = STR$(Number1 * Number2)
Answer$ = LTRIM$(Answer$)
```

and finally the output batch file is created using code identical to ADD.BAS.

The batch file SHOWTIME.BAT uses Batcmd to prompt the user for two numbers, which it stores in the environment by calling SAYERROR.BAT. Next, it uses MULTIPLY.EXE to multiply them together and finally it shows the results. The operational code follows.

```
BATCMD GN Enter First Number To Multiply
CTTY NUL
CALL SAYERROR
CTTY CON
```

```
SET NUMBER1=%ERROR%
BATCMD GN Enter Second Number To Multiply
CTTY NUL
CALL SAYERROR
CTTY CON
SET NUMBER2=%ERROR%
MULTIPLY
CALL JUNK.BAT
DEL JUNK.BAT
ECHO %NUMBER1% X %NUMBER2% = %ANSWER%
SET NUMBER1=
SET NUMBER2=
SET ANSWER=
SET ERROR=
GOTO END
```

Notice the use of `CTTY NUL` and `CTTY CON` to keep SAYERROR.BAT from displaying its message to the screen. MULTIPLY.BAS, MULTIPLY.EXE and SHOWTIME.BAT are all on the disk in the subdirectory \BAT\QB-TIMES. That version of SHOWTIME.BAT has documentation and offers command line help. For SHOWTIME.BAT to work, both BATCMD.EXE and SAYERROR.BAT must be in the current subdirectory or the path.

Builder & Builder Lite

Builder and Builder Lite were covered in Chapter 27. They make the perfect illustration for writing batch file utilities because they can set errorlevel and write to the master copy of the environment. We will look at the same two batch file utilities we looked at under QuickBASIC to see how to write them using Builder Lite. The identical code would work under Builder.

Adding 1 to command line number

Unlike QuickBASIC, Builder Lite requires that we "declare" our variables before we use them. Declaring the variables tells Builder Lite what type of variable they are and reserves a memory slot for them. The commands to declare our variables follow:

```
STRING  Str1, Str2
INTEGER Number, StrLong, Num1, Num2
```

Builder Lite does not work with non-integer numbers, and variables declared as "Integer" automatically ignore the decimal portion of a number. As a result, Builder Lite automatically restricts our answers to integer (whole number) answers.

Next, we need to read in the first input value. Builder Lite makes that very easy; the command `Str1 := "%1"` does that without the looping required under QuickBASIC. Now, we can test this to see if help was requested with the commands that follow:

```
IF Str1 == "/?"
 SAY "Reads A Number From The Command Line"
 SAY "Adds One To It And Sets Errorlevel Equal To Answer"
 EXIT 0
END
IF Str1 == "?"
```

```
  SAY "Reads A Number From The Command Line"
  SAY "Adds One To It And Sets Errorlevel Equal To Answer"
  EXIT 0
END
```

The EXIT 0 command causes Builder Lite to exit the program and set the errorlevel to zero. The SAY command is very similar to the PRINT command in QuickBASIC.

The next error check is to see if the user forgot to enter the number to add one to. The code to do that follows:

```
  IF Str1 == ""
   SAY "No Number Entered On Command Line"
   BEEP
   EXIT 255
  END
```

Because this is an error, the program beeps the speaker with the BEEP command and exits with errorlevel set to 255.

Setting the errorlevel to 255 for an error makes testing for errors in a batch file much easier. Because the IF ERRORLEVEL test is a greater-than-or-equal test, you have to use dual IF-tests to test for a single errorlevel value. However, because errorlevel cannot exceed 255, the single IF ERRORLEVEL 255 test can test for this value.

The next error checking is to loop through the string variable Str1 and make sure that it contains only numbers. The code to do this follows:

```
  Num1 := 0
  REPEAT StrLong
   Num1 := Num1 + 1
   Str2 := MidStr Str1, Num1, 1
   Num2 := StrToNum Str2
   IF Num2 == 0
   IF Str2 <> "0"
      SAY "Non-Numeric Data Entered"
      BEEP
      EXIT 255
   END
   END
  END
```

The REPEAT-loop loops through the string while the MidStr function grabs one character at a time. That single character is converted to a number. If the number is 0, then the single character is checked to see if it is a 0. If not, the user entered a non-numeric input, and the program exits with an error message, sets the errorlevel to 255, and beeps the speaker. This is a more detailed test than was the test in the QuickBASIC program because every character is examined in detail. As a general rule, this is the level of detail you want for your batch file utilities to completely isolate the batch file utility and make sure no errors reach it.

Next, the program performs the actual mathematics with the commands that follow:

```
Number := StrToNum Str1
Number := Number + 1
```

Because the value is to be passed back to the batch file via errorlevel, the program checks to make sure the value is not too high for errorlevel with the code that follows:

```
IF Number > 255
 SAY "Answer Exceeds 255 And "
 SAY "Errorlevel Will Not Handle This"
 BEEP
 EXIT 255
END
```

Because a negative number would not get past the routine that checks for non-numeric data, the program does not now need to check for a negative answer. In MULTIPLY.BLD, this filter is modified to allow negative numbers to pass. Finally, it exits and stores the answer in errorlevel with the command `EXIT Number`.

SHOWADD.BAT has been slightly modified to work with this new version of ADD.EXE. Because the answer is passed back via errorlevel, SHOWADD.BAT now calls on SAYERROR.BAT to convert the errorlevel to an environmental variable named ERROR. Everything is stored on the disk in the \BAT\BLD-ADD subdirectory.

Multiplying two numbers together

Like ADD.BLD, MULTIPLY.BLD must declare variables and check the command line for a help request. Other than using more variables that ADD.BLD, these are performed in an identical fashion.

Next, the two environmental variables NUMBER1 and NUMBER2 must be read into the program. The commands to do this are:

```
Str1 := "%NUMBER1%"
Str2 := "%NUMBER2%"
```

It checks to make sure that NUMBER1 exists by making sure that Str1 has a value. The code that does this follows:

```
IF Str1 == ""
 SAY "No Number Entered In NUMBER1 In Environment"
 BEEP
 EXIT 255
END
```

Str2 is checked in an identical fashion.

Now, each of the two string variables must be checked to see if they contain any non-numeric characters. The code to do that on Str1 follows:

```
StrLong := Length Str1
Num1 := 0
REPEAT StrLong
 Num1 := Num1 + 1
 Str3 := MidStr Str1, Num1, 1
```

```
Num2 := StrToNum Str3
IF Num2 == 0
Flag := 1
IF Str3 == "0" Then Flag := 0
IF Str3 == "-" Then Flag := 0
IF Str3 == "." Then Flag := 0
IF Flag == 1
   SAY "Non-Numeric Data In NUMBER1 In Environment"
   BEEP
   EXIT 255
END
END
END
```

This code loops through all the characters in Str1. It first sets the flag variable Flag equal to one. Next, it pulls off the next character and stores it in the variable Str3 then converts Str3 to a number, Num2. If this number is a 0, then error checking is required to make sure it's a valid number; otherwise, it's a number and no additional error checking is required for this character.

When Num2 equals 0, the code checks to see if the string Str3 is a 0, negative sign or period. If it is, it's a valid character and the Flag variable is changed to 0. Next, the Flag variable is checked to see if it still equals 1. If so, it displays an error message, beeps the speaker and exits the program because a non-valid character was found. Next, the other input is checked in the same way.

After this, we know we have two valid numbers stored in the string variables. They are converted to numbers, multiplied together, and the answer converted back to a string with the code that follows:

```
Number1 := StrToNum Str1
Number2 := StrToNum Str2
Answer := Number1 * Number2
Str1 := IntToStr Answer
```

While the period was allowed as a valid character in the error checking just shown, Builder Lite (and Builder) only works with integers; the process of converting the string to a number strips off the period and any numbers that follow it, leaving the final answer as an integer. It actually strips off the decimal portion rather than rounding, so both 12.001 and 12.999 become 12 when stored to a numeric variable.

The answer is converted back to a string because the output routine requires a string. Finally, the answer is written to the master copy of the environment with the command `SET "Answer" = Str1`.

The batch file SHOWTIME.BAT illustrates MULTIPLY.EXE by prompting the user for two single digits using Batcmd, storing these digits in the environment using SAYERROR.BAT, and then using MULTIPLY.EXE to compute the answer. The operational code is shown here:

```
BATCMD GN Enter First Number To Multiply
CTTY NUL
CALL SAYERROR
CTTY CON
```

```
SET NUMBER1=%ERROR%
BATCMD GN Enter Second Number To Multiply
CTTY NUL
CALL SAYERROR
CTTY CON
SET NUMBER2=%ERROR%
MULTIPLY
ECHO %NUMBER1% X %NUMBER2% = %ANSWER%
SET NUMBER1=
SET NUMBER2=
SET ANSWER=
SET ERROR=
```

The files MULTIPLY.BLD, MULTIPLY.EXE and SHOWTIME.BAT are all on the disk in the subdirectory \BAT\BLD-TIME. That version of SHOWTIME.BAT has documentation and offers command line help.

Turbo Pascal

Many programming languages, such as Turbo Pascal, Turbo C, and QuickBASIC include libraries of routines that enable you to recall the contents of environment variables. By contrast, these same language implementation lack routines that write to new or existing environment variables in the master and/or current environment. The reason for these missing routines is probably the complicated nature of the process of writing to environment variables. The difficulty lies in the fact that, to write to a new or an existing environment variable, you need to manage the list of environment variables maintained by DOS. In order to do this, you must first locate it in the computer's memory, a not too obvious task to begin with. Once you locate the list, you need to perform the following general steps:

1. Determine whether or not the targeted environment variable already exists.
2. If the environment variable exists, you must first remove it from the list.
3. Insert the name of the targeted environment variable and its contents in the list.

This process involves the use of dynamic lists and employs pointers to alter rather critical data in memory—not a programming job for just anyone.

The EnvLib Library Unit

Rather than reinventing the wheel, let me present a library unit that was developed by Sunny Hill Software and is currently sold by TurboPower Software as part of their Turbo Professional package. These routines are provided for your inspection on the disk in the file ENVLIB.PAS. I am providing the source code with the permission of TurboPower Software. This file gives you (and/or your colleagues who program in Turbo Pascal) the opportunity to study how to write routines to manipulate the environment variables of the master and current environment.

Instead of going through the complicated tasks of the various library routines, let me focus on the following few components: the record type EnvRec, procedure MasterEnv, procedure CurrentEnv, function GetEnvStr, and function SetEnvStr.

The EnvRec record type is a structure that stores the segment address location, size, and pointer to the master or to the current environment. The Master and CurrentEnv procedures return information via the parameter Env (which has the type EnvRec). The data in the parameter contains the handle of the targeted environment. The code is shown here:

```
procedure MasterEnv(var Env : EnvRec);
procedure CurrentEnv(var Env : EnvRec);
```

The function GetEnvStr returns the contents of an environment variable. If the targeted environment variable does not exist, the function returns an empty string. The first parameter has the EnvRec record type and is a handle to the targeted environment. The second parameter is a string that represents the name of the accessed environment variable. The code follows:

```
function GetEnvStr(Env : EnvRec; Search : string)
                : string;
```

The function SetEnvStr is perhaps the most valuable library function for our purposes. This function empowers your Turbo Pascal utilities to write a string to new or existing environment variables. The function returns the logical True value if it is successful (that is, if you did not run out of environment space). Otherwise, the function yields False. The first parameter has the EnvRec record type and acts as a handle to the targeted environment. The second parameter is a string that represents the name of the targeted environment variable. The third parameter represents the new value of the targeted environment variable. The code is shown here:

```
function SetEnvStr(Env : EnvRec; Search, Value :
string)
                : Boolean;
```

Now let's look at a few examples that use the EnvLib library units to write Turbo Pascal batch file utilities and test batch files. This batch file utility performs the same tasks as the ADD.BAS batch file utility that I presented earlier in this chapter. The ADD.PAS utility follows the same general program logic in its QuickBASIC counterpart. As you might expect, the differences between the QuickBASIC and Turbo Pascal languages affect the differences between the exact program statements in the ADD.BAS and ADD.PAS utilities.

Adding 1 to command line number

ADD.PAS uses the {$X+} compiler directive to allow the program to discard the result of functions. I use this directive specifically to discard the results of function SetEnvStr. It also contains the Uses clause "Uses EnvLib;" which enables it to import various routines from the EnvLib library unit.

The ADD.PAS program declares a number of variables that store and manipulate data using the code shown below. I will cover these variables as I cover the different blocks of statements in ADD.PAS.

```
VAR Answer, Build : STRING;
 ErrorCode : INTEGER;
 Number : LONGINT;
 E : EnvRec;
```

The first program statement invokes the procedure MasterEnv, which is exported by the EnvLib library unit. The argument for the MasterEnv is the record variable E. This statement obtains vital information related to the location and size of the master environment. The code is shown below:

```
{ Get Information on Master Environment }
 MasterEnv(E);
```

The following statement enables the program to retrieve the first command line argument, using the predefined Turbo Pascal function ParamStr:

```
{Get The First Command-Build Argument }
 Build := ParamStr(1);
```

The argument of "1" selects the first command line argument by number. If there are no command line arguments, the ParamStr function returns an empty string, which is assigned to the string variable Build. Next, the program checks for a help request using an IF-THEN statement. This statement displays on-line help and ends the program when the variable Build contains either an empty string, the string "/?", or or the string "?." The code for this is shown here.

```
{ Now, Check For Help Request }
 IF (Build = '/?') OR (Build = '?') OR (Build = '') THEN
BEGIN
 WRITE('Adds One To Number Passed ');
 WRITELN('And Saves It To Variable ANSWER. ');
 WRITE('A Batch File Should Examine The ');
 WRITELN('The Environment Variable ANSWER.');
 HALT;
 END;
```

If the program does not execute the statements in the THEN clause, it proceeds to convert the characters in variable Build into a number. That number is stored in the long integer variable Number. Because of the extended range of long integers, you can use the Turbo Pascal utility to add 1 to the ANSWER environment variable when the latter already has values well above 32767—which is the upper limit in the QuickBASIC and Builder Lite version. This extended numeric range feature is one of the benefits that come from using a programming language like Turbo Pascal.

The conversion statement uses the predefined Turbo Pascal procedure VAL and employs the variable ErrorCode to report the conversion error-status. The ErrorCode is an integer variable that stores the index of the first offending character in the string of variable Build. If there is no error in the conversion, the variable ErrorCode contains 0. Unlike the QuickBASIC VAL function, the Turbo Pascal version either entirely succeeds or fails in its task—there are no results from partial conversion.

```
{Now Convert Build To Number }
 VAL(Build, Number, ErrorCode);
```

The program then checks the conversion error status by using an IF-THEN-ELSE statement, as shown next. This statement determines if the variable ErrorCode contains a positive value. If this condition is true, there is a conversion error and the program executes the statements in the THEN clause. These statements display a warning message and end the program. Otherwise, the program executes the statement in the ELSE clause to increment the content of variable Number.

```
IF ErrorCode > 0 THEN BEGIN
 WRITELN('Invalid Input');
 WRITELN('Enter ADD /? For Help');
 WRITE(^G); { Beep }
```

```
    HALT;
    END
    ELSE
    Number := Number + 1;
```

Following a successful addition, the program then converts the value in variable Number to a string and stores it in variable Answer. The program uses the predefined Turbo Pascal procedure STR to perform this task, as shown here.

```
{Convert Number To String To
 Avoid Space In Batch File   }
 STR(Number, Answer);
```

The last statement saves the value of string Answer in the environment variable ANSWER. This statement uses the SetEnvStr procedure which is exported by the EnvLib library unit. Notice that the first argument in the code below for the SetEnvStr procedure is the record variable E. The second argument is the string "ANSWER" which specifies the name of the environment variable. The third argument is the variable Answer which supplies the value written to the environment variable ANSWER.

```
{ Assign String to Environment Variable ANSWER }
 SetEnvStr(E, 'ANSWER', Answer);
```

The Pascal version of SHOWADD.BAT is very similar to its QuickBASIC counterpart and executes a batch file loop for 20 times. The operational code is shown next. The main difference between the two versions is the loop that invokes the ADD.EXE utility.

```
:TOP
 ADD %Answer%
 ECHO Loop Number %ANSWER%
 IF NOT %ANSWER%==20 GOTO TOP
 ECHO Loops Done
GOTO END
```

Notice that, after the loop calls the ADD.EXE utility, there is no call to a temporary batch file. Because ADD.EXE writes directly to the environment variable ANSWER, there is no need for such a batch file. Consequently, the Turbo Pascal version of SHOWADD.BAT runs faster than the QuickBASIC version because it bypasses the invocation of other batch files.

The files ADD.PAS, ADD.EXE and SHOWADD.BAT are all on the disk in the subdirectory \BAT\PAS-ADD. That version of SHOWADD.BAT has documentation and offers command line help.

Adding 1 & letting the user pick the environmental variable

The ADD.PAS batch file utility systematically stores the result of the addition in the environment variable ANSWER. We can modify the utility to store the result in a environment variable which the user specifies. We can also make the new version use the environment variable ANSWER if the user does not enter a name.

The PLUS.PAS batch file utility will be this enhanced version of ADD.PAS. The new utility uses the variable EVarName to store the name of the custom environment variable. After assigning the first command line argument to variable Build, the program assigns the second command line argument to the variable EVarName.

This argument represents the name of the targeted environment variable and the code is shown next:

```
{Get The Second Command-Build Argument }
 EVarName := ParamStr(2);
```

The program then uses an IF statement to assign the string "ANSWER" to the variable EVarName if there is no second command line argument. The IF statement compares the variable EVarName with an empty string to test whether or not the above condition is true, using the code shown next:

```
{Use The Name ANSWER If User Does Not
 Specify The Name Of An Environment Variable }
 IF EVarName = '' THEN
 EVarName := 'ANSWER';
```

The rest of the lines in PLUS.PAS are the same as the ones in ADD.PAS, except for the last statement, as shown next. The call to the SetEnvStr uses a different second argument. The PLUS.PAS program passes the variable EVarName to the SetEnvStr procedure to supply it the name of the targeted environment variable.

```
{Assign String to Environment Variable
 Specified By The Pascal Variable EVarName  }
 SetEnvStr(E, EVarName, Answer);
```

The files PLUS.PAS, PLUS.EXE and SHOWPLUS.BAT are all on the disk in the subdirectory \BAT\PAS-PLUS. SHOWPLUS.BAT is identical to SHOWADD.BAT except that the environmental variable "sum" is specified on the command line to replace "answer" and the "PLUS" command is used instead of the "ADD" command.

Multiplying two numbers together

Let's look at another Turbo Pascal batch file utility. This one presents the Pascal version of the MULTIPLY.BAS utility that I presented earlier. The utility carries out the following tasks:

1. Reads the contents of two environment variables NUMBER1 and NUMBER2.
2. Converts the contents of NUMBER1 and NUMBER2 into numbers.
3. Multiplies the converted numbers.
4. Writes the result in the environment variable ANSWER.

Like the ADD.PAS utility, the MULTIPLY.PAS imports from the EnvLib library the routines it needs to manipulate the environment variables using the command "Uses EnvLib;".

The MULTIPLY.PAS utility also declares a set of variables to store and manipulate data using the commands:

```
VAR Answer, Build, Input1, Input2 : STRING;
 ErrorCode1, ErrorCode2 : INTEGER;
 Number1, Number2 : LONGINT;
 E : EnvRec;
```

The first program statement invokes the procedure MasterEnv (exported by the EnvLib library unit) to store the data for the master environment in the record variable E, using the commands:

```
{ Get Information on Master Environment }
 MasterEnv(E);
```

Next, the program assigns the first command line argument to the variable Build. This command line argument serves to invoke the on-line help. Otherwise, the MULTIPLY.PAS utility needs no command line arguments because it handles its input and output data through the environment variables NUMBER1, NUMBER2, and ANSWER. That code is shown next:

```
{ Get The First Command-Build Argument }
 Build := ParamStr(1);
```

The program then checks if the utility is invoked with a help request. The program performs this task using the following IF-THEN statement. Notice that the statement examines the condition where the variable Build stores either string "/?" or "?." The condition does not include comparing variable Build with an empty string because the utility works without command line arguments. This code is shown next:

```
{ Now, Check For Help Request }
 IF (Build = '/?') OR (Build = '?') THEN BEGIN
 WRITE('Multiplies Two Numbers Together ');
 WRITELN('Passed To It Via The Environment');
 WRITE('Using The Names NUMBER1 And NUMBER2.');
 WRITELN('Assigns Result To ANSWER.');
 WRITE('A Batch File Should Examine The ');
 WRITELN('The Environment Variable ANSWER.');
 HALT;
 END;
```

In the next step, the program copies the contents of environment variables NUMBER1 and NUMBER2 to the string variables Input1 and Input2, respectively. This step involves the GetEnvStr function, which is exported by the EnvLib library unit. The first argument for the GetEnvStr function is the record variable E. The second argument is the name of the accessed environment variable. The GetEnvStr function yields the contents of the specified environment variable, or an empty string if the variable is undefined. That code follows:

```
{ Now Get The Numbers From The Environment }
 Input1 := GetEnvStr(E, 'NUMBER1');
 Input2 := GetEnvStr(E, 'NUMBER2');
```

It is possible that either or both NUMBER1 and NUMBER2 are undefined. The program detects the existence of the environment variables NUMBER1 and NUMBER2 by examining the contents of variables Input1 and Input2. The utility uses two similar IF-THEN statements that determine if either variable Input1 or Input2 stores an empty string. If so, the program displays a warning message and then exits. That code is shown next for Input1:

```
IF Input1 = '' THEN BEGIN
 WRITELN('NUMBER1 Environmental Variable Missing');
 WRITE(^G); {.Beep }
 HALT
 END;
```

While not shown, the code for Input2 is identical except for the variable names.

Once the program determines that the environment variables NUMBER1 and NUMBER2 exist, it proceeds with converting the contents of the variables Input1 and Input2 into numbers. The program uses the predefined VAL procedure to store the converted numbers in the long integer variables Number1 and Number2. The VAL procedures use the variables ErrorCode1 and ErrorCode2 to store the error status of the conversions. That code follows:

```
{ Now Convert Strings To Number }
 VAL(Input1, Number1, ErrorCode1);
 VAL(Input2, Number2, ErrorCode2);
```

Next, the program determines whether or not both of the conversions went smoothly. The program uses two similar IF-THEN statements that display error messages and exit if the values in either ErrorCode1 or ErrorCode2 are positive. That code is shown next for ErrorCode1:

```
{ Now Check To Make Sure That
 There Is No Conversion Error }
 IF ErrorCode1 > 0 THEN BEGIN
 WRITE('Invalid Input For NUMBER1 ');
 WRITELN('Environmental Variable');
 WRITELN('Enter MULTIPLY /? For Help');
 WRITE(^G);
 HALT;
 END;
```

While not shown, the code for NUMBER2 is identical except for the variable names.

After the just-shown set of verifications, the program proceeds to multiply the numbers in variables Number1 and Number2 and store the string image of the result in the variable Answer. The program performs this two tasks in a single call to procedure STR, shown here:

```
{ Convert Number To String To Avoid
 Space In Batch File              }
 STR(Number1 * Number2, Answer);
```

The last statement saves the value of string Answer in the environment variable ANSWER. This statement uses the SetEnvStr procedure, which is exported by the EnvLib library unit. The statement is similar to the last one in the ADD.PAS utility. That code follows:

```
{ Assign String to Environment Variable ANSWER }
 SetEnvStr(E, 'ANSWER', Answer);
```

The files MULTIPLY.PAS, MULTIPLY.EXE and SHOWTIME.BAT are all on the disk in the subdirectory \BAT\PAS-TIME. The code for SHOWTIME.BAT is identical to the code for the QuickBASIC batch file.

Windows 3.1

Windows 3.1 gives you the option of shelling out to DOS by running another copy of COMMAND.COM. This is the same action it takes if you run a batch file. The Builder Lite and Pascal batch file utilities that write directly to the master copy of the environment behave differently under Windows.

When either utility runs, there are four different environments being used by DOS:

1. The master copy of the environment created by DOS when the computer booted.
2. The copy of the environment passed by DOS to Windows.
3. The second master copy of the environment created by COMMAND.COM when Windows ran it to give you a DOS prompt or run a batch file.
4. The copy of this second master copy of the environment passed to either the Builder Lite or the Pascal routine.

Just keeping up with all these copies of the environment is difficult!

When Builder Lite reads from or writes to the "master" copy of the environment, it uses line three above, the second copy of the master environment created when COMMAND.COM ran Windows. That way, the changes it makes are available to the batch file while it is running. Most of the time this is what you want the utility to do.

When a Pascal program reads from or writes to the "master" copy of the environment, it uses the true master copy of the environment, or number 1 earlier. That means if the environmental variables NUMBER1 and NUMBER2 did not exist prior to loading Windows, then MULTIPLY.EXE will report an error even if the batch file that runs it under Windows creates them—because that batch file creates them in the COMMAND.COM environment running under Windows (number 3 earlier). It also means that when MULTIPLY.EXE writes the results to ANSWER while running under Windows, that value will not be available to a batch file because the batch file reads the environment from the version of COMMAND.COM that Windows ran (number 3 earlier) and not the master copy of the environment (number 1 earlier) modified by MULTIPLY.EXE.

In other words, these Pascal utilities cannot be used while running under Windows while the Builder Lite utilities can. As a result, it is very important to test your utilities under Windows and understand how they interact with Windows if you ever intend to use them with Windows.

Summary

A batch file utility is any program designed to take information from a batch file or provide information to the batch file in order to enhance its operation. Because they generally only perform one specific task, batch file utilities tend to be some of the simplest programs you can write in any language.

In order to program a batch file utility, you need a way to get information into the program and a way of getting information out of the program.

There are four ways to get information into a batch file utility: the command line, environment, ask the user, or poll the equipment.

There are four ways to get information out of a batch file utility and back into the batch file: errorlevel, the master copy of the environment, an on-the-fly batch file, and printing to the screen or printer.

The QuickBASIC compiler lacks the ability to write to the master copy of the environment or to set the errorlevel when a program exits.

The COMMAND$ function lets QuickBASIC read the command line sans the name of the program.

Once you obtain the command line in QuickBASIC with the COMMAND$ function, you have to manually separate it into its component parts in order to access the different "replaceable parameters" separately.

In QuickBASIC, the "END" command causes the program to immediately terminate and return control to DOS. After displaying help, this is an excellent way for help routines to terminate.

In QuickBASIC, numeric results have to be converted to strings and trimmed to avoid placing spaces in the environment.

In QuickBASIC, the only viable output method for a batch file utility is writing a temporary batch file for the main batch file to CALL to place the results into the environment.

Batch file utilities need extensive error checking to make sure the user entered exactly the information that the utility expects to receive.

Even when a batch file is taking its usual input from the environment, user, or by polling the equipment, it still must check the command line to see if the user requested help.

In QuickBASIC, the ENVIRON$ function is used to read in environmental variables. All environmental variables come in as string variables even if they are just numbers.

Builder and Builder Lite can read from the command line and environment and can set errorlevel and write to the master copy of the environment. This makes these languages excellent tools for writing batch file utilities.

Builder Lite uses %1 to represent the first replaceable parameter, %2 for the second, and so on, so it is very easy to read replaceable parameters into a Builder Lite program.

The `EXIT #` command causes a Builder Lite program to immediately exit to DOS and set the errorlevel to the number specified after the command.

Setting the errorlevel to 255 for an error makes testing for errors in a batch file much easier.

Builder Lite can read in environmental variables by surrounding their name with percent signs and double quotes so the command `Str1 := "%NUMBER1%"` reads the environmental variable NUMBER into the variable Str1.

Builder Lite can write to the master copy of the environment with a command like `SET "Answer" = Str1`.

The Turbo Professional package, developed by Sunny Hill Software and marketed by TurboPower Software, allows Turbo Pascal programs read from and write to the master copy of the environment.

The function SetEnvStr in the Turbo Professional package is used by a Turbo Pascal program to write a string to a new or existing environmental variable.

Sometimes when storing results in an environmental variable, it is useful to let the user specify the name of the environmental variable to use.

When a Builder Lite program alters the "master" copy of the environment under Windows, it alters the COMMAND.COM copy of the environment for that DOS session rather than the true master copy of the environment.

When a Pascal program alters the "master" copy of the environment under Windows, it alters the true master copy of the environment. This can cause batch files running under Windows to bomb if they depend on a Pascal utility.

35 Modifying DOS with DOSKEY & batch files

If you've upgraded to DOS 5 or 6, then you have a copy of DOSKEY. DOSKEY is a memory resident program that allows you to change the operation of any of the DOS commands. Let's begin by seeing just how powerful DOSKEY is.

If you do not currently load DOSKEY, change to your DOS subdirectory and enter `DOSKEY` at the command line. Now, change to your batch subdirectory and create this simple batch file:

```
@ECHO OFF
FOR %%J IN (%1) DO ECHO %%J...To Be Deleted
BATCMD YN Do You Wish To Delete These Files?
IF NOT ERRORLEVEL 1 GOTO END
DEL %1
:END
```

Give the batch file the name of KEYDEL.BAT. Since KEYDEL.BAT uses Batcmd, it will need to be in the path.

Now, create a temporary subdirectory and create some temporary files in that subdirectory called JUNK.1 through JUNK.9. Now, enter the command `DOSKEY DEL=KEYDEL $1` at the command line. Now, enter the command `DEL JUNK.?` at the command line. As you can see, DOS runs your macro rather than using its internal DEL command.

If you look back at KEYDEL.BAT, you will see that it uses the DEL command to perform the actual deletion, yet DOSKEY is supposed to replace the DEL command with KEYDEL.BAT! That is the real advantage of DOSKEY. When you change the command with DOSKEY, it only changes for the command line, so your batch files continue to behave normally. So when you enter the DEL command from the

command line, DOSKEY maps it onto KEYDEL.BAT; but when a batch file uses the DEL command—including KEYDEL.BAT—it is not mapped, so it accesses the actual DOS internal command.

Before we continue, I need to let you know about one problem. There is a bug in some versions of DOSKEY that prevents the /P switch from working when you alias a command. No matter if it is built into the batch file or passed to the batch file as a parameter, the batch file will abort the first time the user answers yes to the prompt and the file will not be erased.

Later on in this chapter, we will see how to get around that with a clever batch file. But first, let's see more about how DOSKEY operates.

DOSKEY basics

Because you read this far in this book, it's a safe bet you are very familiar with batch files. Well, a DOSKEY macro is very much like a batch file. However, some of the symbols you use are different. Those differences include:

$1–$9	In a batch file, the replaceable parameters are represented by 0–%9, where %0 is the name of the batch file and %1–9 are the first nine replaceable parameters entered on the command line after the name of the batch file. In a DOSKEY macro, the name of the macro is not available and the replaceable parameters are represented by $1–$9 rather than %1–%9; however, they work the same.
$*	This stands for the entire command line after the macro name, making it easy to pass long command lines through DOSKEY.
Piping	You cannot include DOS piping in a DOSKEY command directly because DOS processes the pipe before it gets to DOSKEY. To overcome that, DOSKEY allows you to replace the pipes with symbols not processed by DOS. The greater-than pipe (>) becomes $G, the less-than pipe (<>) becomes %L, and the vertical bar pipe (¦) becomes $B. However, DOSKEY still has problems with pipes so that many times a properly written macro containing pipes executes but the pipe does not work.

You make a DOSKEY assignment with a command like

```
DOSKEY Command=Replacement <parameters>
```

where *command* is the item you will be typing on the command line, *replacement* is what you want DOSKEY to replace it with and *parameters* are any parameters you want DOSKEY to pass from the command line to the replacement command.

What can I do with DOSKEY?

DOSKEY has three different functions, command shortening, command name changing and function changes. Let's take a look at how each of these works.

Shortening commands

This is actually the easiest way to use DOSKEY. You simply define the "shorthand" you plan to use on the command line. For example, if you were cleaning up your hard disk and planned on repeatedly entering a directory command, you might use the following DOSKEY command

```
DOSKEY D=DIR $1 /P /OE
```

That way, if you entered D on the command line by itself, you would get a directory of all the files in the current subdirectory. Entering `D *.TXT` would give you just the text files.

Name changing

This allows you to change the names of internal commands or programs to something the user is more likely to remember. For example, you might use the following to make a system more friendly:

```
DOSKEY KILL=DEL $* /P
DOSKEY GONE=DEL $* /P
DOSKEY DESTROY=DEL $* /P
```

This allows users to use commands they are more likely to remember to perform common tasks. It also illustrates that the term "command name changing" is a little misleading. While it does allow you to access the command under a new name, or even several new names as shown previously, the old name also works.

Function change

We've already seen this one in action with KEYDEL.BAT. Here, you write a batch file (or even a .EXE or .COM program) to perform a task in place of a DOS internal command and then use DOSKEY to map the DOS internal command onto the batch file.

Occasionally, you might have a command remapped using DOSKEY but want to access the original command. For example, you might have the DEL command remapped to execute a batch file. That is very easy to do. Just enter a space on the command line before the command. Spaces before commands does not affect DOS, but it fools DOSKEY's remapping so the command works without being remapped.

Putting it together

If you loaded DOSKEY as described above, it loaded with the default size of 512 bytes for its storage area. While 512 bytes is large enough to demonstrate the power of DOSKEY, it is too small to be really useful. To increase the size of the DOSKEY storage space, load it with a command like this:

```
DOSKEY /BUFSIZE=2000
```

The 2000 is the size (in bytes) that DOSKEY allocates for storage. This space is used both to store macros and old command lines. I use 2000 on my system, and I rarely run out of space.

If you've been following along with this chapter by working with your computer, then you've already loaded DOSKEY with the default buffer size. That's too small for what we will be doing next. DOSKEY lacks an unload command so you must reboot to remove it from memory. However, you can load a second copy and give that second copy a larger storage space. To do that, enter the following command:

```
DOSKEY /REINSTALL BUFSIZE=2000
```

To show you the power of DOSKEY, run KEYASSGN.BAT in FIG. 35-1. This batch file creates a number of new commands and modifies the way several commands operate. TABLE 35-1 summarizes these new commands.

Several of those new commands run one of two batch files, KEYERASE.BAT in FIG. 35-2 and MOVEIT.BAT in FIG. 35-3. The commands relying on KEYERASE.BAT just change the way existing commands work, while the commands relying on MOVEIT.BAT create entirely new commands. You might also want to notice how KEYERASE.BAT avoids the problem associated with using the /P switch in a DOSKEY macro.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: KEYASSGN.BAT`
`REM PURPOSE: Make DOSKEY Assignments`
`REM VERSION: 1.00`
`REM DATE: December 25, 1991`
`REM Batch Assignment`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`REM Batch Files` `REM ==========`	Documentation remarks.
`ECHO Attaching Batch Files` `ECHO Please Wait`	This batch file takes several seconds to run with nothing showing on the screen to tell the user it's running.
`DOSKEY DEL=KEYERASE $*` `DOSKEY ERASE=KEYERASE $*`	Remap the *DEL* and *ERASE* commands to KEYERASE.BAT.

`DOSKEY NUKE=KEYERASE $*`
`DOSKEY GONE=KEYERASE $*`
`DOSKEY KILL=KEYERASE $*`
`DOSKEY ZAP=KEYERASE $*`
	Remap four new keywords to KEYERASE.BAT. Since they are remapped to the same batch file, these new commands will function identically to the *DEL* and *ERASE* commands.

`DOSKEY MOVE=MOVEIT $*`
`DOSKEY RELOCATE=MOVEIT $*`
`DOSKEY NEWDIR=MOVEIT $*`
	Create three new commands by mapping them onto a batch file. If only one command accessed this batch file, it would have been easier just to give the batch file the name of the new command and include it in a subdirectory in the path. Since these three names access the same batch file, they will function identically.

`REM Commands` `REM ========`	Documentation remarks.

`BATCMD SL`
`ECHO Renaming Commands`
`ECHO Please Wait`
	Skip a line and let the user know the batch file is still running.

`DOSKEY CHECK=CHKDSK $*`	Remap the *CHECK* command to run CHKDSK. You could write a batch file called CHECK.BAT to do the same but this approach uses no disk space.
`DOSKEY MACROS=DOSKEY /MACROS` `DOSKEY MACRO=DOSKEY /MACROS`	Remap these two commands onto the DOSKEY command to show the macros that have been defined.

35-1
KEYASSGN.BAT creates a number of useful DOSKEY commands.

Batch File Line	Explanation
`DOSKEY DIR=DIR $1 /P`	Alter the way the *DIR* command operates by forcing it to include a /P switch.
`DOSKEY FILES=DIR $1 /P`	Add a new command that operates by running *DIR*.

`DOSKEY FF1=ECHO ^L $G LPT1`
`DOSKEY FF2=ECHO ^L $G LPT2`
`DOSKEY FORMFEED1=ECHO ^L $G LPT1`
`DOSKEY FORMFEED2=ECHO ^L $G LPT2`
	Create several formfeed commands that pipe a ^L (formfeed) character to the printer.

`DOSKEY HIDE=ATTRIB +H $1`	Replace the hard to understand *ATTRIB +H* command with an easy-to-understand replacement.

`DOSKEY UNHIDE=ATTRIB -H $1`
`DOSKEY READONLY=ATTRIB +R $1`
`DOSKEY RO=ATTRIB +R $1`
`DOSKEY UNREADONLY=ATTRIB -R $1`
`DOSKEY UNRO=ATTRIB -R $1`
	Replace several other *ATTRIB* commands with easy-to-understand and easy-to-remember replacements.

`DOSKEY LINES25=MODE CON: LINES=25`
`DOSKEY LINES43=MODE CON: LINES=43`
`DOSKEY LINES50=MODE CON: LINES=50`
	Add commands to change the display type. These commands require that you load ANSI.SYS in your CONFIG.SYS file.

`DOSKEY LIST=FOR %%J IN ($1) DO TYPE` `  %%J $b MORE`	Create a command to display text files using DOS commands.
`DOSKEY FILEFIND=ATTRIB \$1 /S`	Create a command to locate files anywhere on the current drive using DOS commands.

`BATCMD SL`
`ECHO Finished`
`GOTO END`
	Tell the user the batch file is finished and exit the batch file.

`:HELP`
`  ECHO This Batch File Uses DOS 5.0's`
`  ECHO DOSKEY To Make A Number Of`
`  ECHO Useful Command Changes`
	First part of section that displays help when the user starts the batch file with a /? or a ? as the first replaceable parameter.

`  IF NOT EXIST KEYASSGN.BAT GOTO END`
`  ECHO Press Any Key To See The`
`  ECHO Assignments`
`  PAUSE`
`  TYPE KEYASSGN.BAT | FIND "DOSKEY "|`
`  FIND "=" | FIND "TYPE`
`  KEYASSGN.BAT"/V | MORE`
`GOTO END`
	This section types the file to display the commands, as long as the file is in the current subdirectory.

`:END`	Label marking the end of the batch file.

35-1
Continued.

Table 35-1

New or Modified Command	Function
DEL	Runs KEYERASE.BAT instead of functioning as an internal command.
ERASE	Runs KEYERASE.BAT instead of functioning as an internal command.
GONE	A new command to erase files.
KILL	A new command to erase files.
MOVE	Moves files to a new location by copying them to the new location and then deleting them from the original location. It has extensive error-checking.
NUKE	A new command to erase files.
ZAP	A new command to erase files.
RELOCATE	Moves files to a new location by copying them to the new location and then deleting them from the original location. It has extensive error-checking.
NEWDIR	Moves files to a new location by copying them to the new location and then deleting them from the original location. It has extensive error-checking.
CHECK	Runs CHKDSK.
MACROS	Displays the currently assigned DOSKEY macros.
MACRO	Displays the currently assigned DOSKEY macros.
DIR	Automatically includes the /P switch.
FILES	Displays a list of files.
FF1	Sends a formfeed to LPT1.
FF2	Sends a formfeed to LPT2.
FORMFEED1	Sends a formfeed to LPT1.
FORMFEED2	Sends a formfeed to LPT2.
HIDE	Changes a file's attribute to hidden.
UNHIDE	Turns off a file's hidden attribute.

Table 35-1 Continued.

New or Modified Command	Function
READONLY	Turns on a file's read-only attribute.
RO	Turns on a file's read-only attribute.
UNREADONLY	Turns off a file's read-only attribute.
UNRO	Turns off a file's read-only attribute.
LINES25	Sets the text mode of the screen to display 25-lines. Requires loading ANSI.SYS in the CONFIG.SYS file.
LINES43	Sets the text mode of the screen to display 43-lines. Requires loading ANSI.SYS in the CONFIG.SYS file. Also requires an EGA screen.
LINES50	Sets the text mode of the screen to display 50-lines. Requires loading ANSI.SYS in the CONFIG.SYS file. Also requires a VGA screen.
LIST	Types files to the screen and pipes the output through the MORE filter.
FILEFIND	Locate files on the current drive only.

35-2
KEYERASE.BAT replaces the DEL and ERASE commands once you have run KEYASSGN.BAT.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: KEYERASE.BAT`
`REM PURPOSE: DOSKEY Erase Replacement`
`REM VERSION: 1.00`
`REM DATE: December 25, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`IF (%1)==(XYZ123ABC) GOTO PROMPT`	When the batch file calls itself as a subroutine, jump to a section to handle that.
`IF (%1)==() GOTO NOTHING`	If the user fails to enter the name of any files to delete, jump to an error-handling section.
`:TOP`	Label marking the top of the main portion of the batch file.
`FOR %%J IN (%1) DO ECHO %%J....` `To Be Erased!`	List the files to be deleted.

Batch File Line	Explanation
`BATCMD YN Do You Want To Erase These Files?`	Ask the user about deleting these files.
`IF NOT ERRORLEVEL 1 GOTO END`	If the users says no, exit the batch file.
`BATCMD YN Do You Wish To Be Ask Individually?`	Ask the user about prompting for each file individually.
`IF NOT ERRORLEVEL 1 DEL %1` `IF NOT ERRORLEVEL 1 GOTO NEXT`	If the user does not want individual prompting, delete the files and jump to a section called NEXT.
`:TOPLOOP`	Label marking the top of a section to individually delete the files. The user only reaches this point by answering yes to the above prompt asking about individual deletion.
`FOR %%J IN (%1) DO CALL KEYERASE XYZ123ABC %%J`	Loop through all the files to be deleted. For each file, call this batch file as a subroutine and pass it a subroutine flag and the name of the batch file to delete.
`GOTO NEXT`	Go to the NEXT section to continue looping.

`:NEXT`
`SHIFT`
`IF NOT (%1)==() GOTO TOP`
`GOTO END`
	Section to check to see if the user specified another set of files to delete. If so, jump back to the processing section of the batch file.

`:NOTHING`
`ECHO Did Not Specify Files To Erase!`
`ECHO The Format For The Command Is:`
`BATCMD SL`
`ECHO ERASE *.BAK`
`ECHO DEL *.TXT`
`BATCMD SL`
`ECHO Where You Replace *.BAK or *.TXT`
`ECHO With The File You Wish To Erase`
`GOTO END`
	Section to handle the problem when the user fails to specify any files to erase.

`:PROMPT`
`BATCMD YN Delete %2?`
`IF ERRORLEVEL 1 DEL %2`
`GOTO END`
	Subroutine section that handles asking about deleting a single file. Note the use of %2, that's because %1 is the subroutine flag.

`:HELP`
`ECHO This Batch File Is An Intelligent`
`ECHO Replacement For The DEL And ERASE`
`ECHO Commands Designed To Be Mapped To`
`ECHO Those Commands Using DOSKEY`
`ECHO OK To Run From Command Line`
`GOTO END`
	Section that displays help when the user starts the batch file with a /? or a ? as the first replaceable parameter.

`:END`	Label marking the end of the batch file.

35-2
Continued.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: MOVEIT.BAT`
`REM PURPOSE: DOSKEY Move Command`
`REM VERSION: 1.00`
`REM DATE: January 12, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`IF (%1)==() GOTO NOSOURCE`	If the user did not enter the name of a file to move, jump to an error-handling routine.
`IF (%2)==() GOTO NOTARGET`	If the user did not enter the name of a subdirectory to move the files to, jump to an error-handling routine.
`IF NOT EXIST %1 GOTO NOFILES`	If the files the user wants to move do not exist, jump to an error-handling routine.
`IF NOT EXIST %2\NUL GOTO DIRGONE`	If the target subdirectory does not exist, jump to an error-handling routine.
`IF EXIST %2\%1 GOTO ALREADY`	If the files already exist in the target subdirectory, jump to an error-handling routine.
`XCOPY %1 %2\*.*`	Copy the files.
`IF ERRORLEVEL 1 GOTO OOPS`	If there is a copy error, jump to an error-handling routine.
`DEL %1`	Delete the original files.
`GOTO END`	Exit the batch file.

`:OOPS`
`BATCMD BE 5`
`ECHO A Copy Error Has Occurred`
`ECHO Source Files *NOT* Erased`
`GOTO END`
	When XCOPY reports an error, display an error-message and exit the batch file.

`:ALREADY`
`ECHO Files Named %2\%1 Already Exist`
`ECHO This Routine Will *NOT*`
`ECHO Overwrite Files`
`ECHO You Must Delete The %2\%1 Files`
`ECHO In Order To Use This Routine`
`GOTO END`
	When the files already exist in the target subdirectory, display an error-message and exit the batch file.

`:DIRGONE`
`ECHO You Specified Moving The %1`
`ECHO Files Into The %2 Subdirectory`
`ECHO However The %2 Subdirectory`
`ECHO Does Not Exist`
`ECHO (Must Be Entered As C:\BAT`
`ECHO *NOT* C:\BAT\)`
`BATCMD SL`
`GOTO SYNTAX`
	When the specified target subdirectory does not exist, display an error-message and jump to a routine to display the proper syntax before exiting the batch file.

35-3
MOVEIT.BAT adds several new commands to DOS, including a MOVE command.

Batch File Line	Explanation
:NOFILES ECHO You Specified Moving The ECHO %1 Files But None Of ECHO These Files Exist GOTO SYNTAX	When the source files do not exist, display an error-message and jump to a routine to display the proper syntax before exiting the batch file.
:NOSOURCE ECHO Source Files Not Specified GOTO SYNTAX	When the user does not specify the source files to move, display an error-message and jump to a routine to display the proper syntax before exiting the batch file.
:SYNTAX ECHO Enter MOVEIT {Source} {Target} ECHO Where {Source} Are The Files ECHO To Move And {Target} Is The ECHO Location To Move Them To BATCMD SL ECHO DOSKEY Users May Enter The ECHO Command As MOVE {Source} {Target} ECHO Although The MOVEIT Command ECHO Will Also Work GOTO END	This section displays the proper syntax for the command and then exits the batch file. It is called by several other error sections.
:NOTARGET ECHO Target Subdirectory Not Specified GOTO SYNTAX	When the user specifies a target subdirectory that does not exist, display an error-message and jump to a routine to display the proper syntax before exiting the batch file.
:HELP ECHO MOVEIT.BAT Moves Files From One ECHO Location To Another With ECHO Extensive Error Checking BATCMD SL ECHO Designed To Work With DOSKEY ECHO However It Can Also Be Run ECHO As A Stand-Along Batch File GOTO END	Section that displays help when the user starts the batch file with a /? or a ? as the first replaceable parameter.
:END	Label marking the end of the batch file.

35-3
Continued.

Command line recall

DOSKEY also stores your command lines so you can reuse them. Just press the up arrow to scroll back through your commands one at a time. When you get to one you want to reuse, just press Return to run it.

If you know how the command started, just type in the first few letters of the command and press F8. That will take you back to the last command that started with the letters you entered. If that is not the command you wanted, press F8 again to cycle back to the command before that starting with the letter you entered.

If you want to review all the commands you have entered, just press F7. It lists all your commands with line numbers to the left of them. If there is more than will fit on one screen, it shows them one screen at a time and waits for you to press a key

before showing the next screen. You can use the numbers it shows to select a command by pressing F9 and entering the number. That brings the command to the command line so you can make sure you want to run it. If you do, just press Return.

I've had it!

It's easy to purge the command line storage and macros from DOSKEY. Pressing Alt-F7 purges all the command lines and pressing Alt-F10 purges the macros. You can purge a single DOSKEY macro with the command:

```
DOSKEY Macro=
```

where *Macro* is the name of the macro to purge and nothing follows the equal sign.

Summary

DOSKEY allows you to have DOS automatically translate commands you enter on the command line into different commands before running them.

DOSKEY also allows you to map commands onto batch files and pass the contents of the command line to that batch file.

Having DOSKEY map a command onto a batch file can cause that command to behave far differently than its normal DOS operation.

DOSKEY allows you to easily recall old command lines.

DOSKEY macros and command lines can easily be purged from memory.

Once you see how powerful DOSKEY is, I'm sure it'll be one of the programs you load in your AUTOEXEC.BAT file!

Part eight

Configuration

When you turn on your computer, after performing a few necessary tasks, DOS processes two user-created files—CONFIG.SYS and AUTOEXEC.BAT. To a great extent, these two files determine the configuration of your computing environment. The CONFIG.SYS file is processed first, before even COMMAND.COM is loaded, while the AUTOEXEC.BAT file is processed last.

CONFIG.SYS under DOS 3.3 This chapter covers the basics of configuration and introduces those commands that were available under DOS 3.3. As a general rule, these configuration commands will run under any version of DOS. All readers will want to examine this chapter because it covers the basic components of the CONFIG.SYS file (such as commands like FILES and BUFFERS). **Chapter 36**

CONFIG.SYS under DOS 5 DOS 5 expanded the CONFIG.SYS file to include memory management commands. This chapter explains how to use those enhancements. **Chapter 37**

CONFIG.SYS under DOS 6 DOS 6 greatly expands the power and usefulness of the CONFIG.SYS file. It now gives you the ability to select among several different configurations when you boot or bypass the CONFIG.SYS file all together. In addition, the CONFIG.SYS file now has the ability to communicate with your AUTOEXEC.BAT file. This chapter explains how to construct a very powerful and useful CONFIG.SYS file under DOS 6. **Chapter 38**

AUTOEXEC.BAT, WINSTART.BAT, & shutdown files The AUTOEXEC.BAT file is designed to set up your computing environment. However, if you write it like any other batch file, you are ignoring some of that power. This chapter shows you how to write your AUTOEXEC.BAT file for maximum power. If there are memory resident programs you only want available while in Windows, then WINSTART.BAT is the spot to load them. Also, while DOS does not offer a shutdown batch file that automatically runs when you turn the computer off, it would be a good idea to have one. This file could do things like park the heads on your computer. This chapter also explores this issue. **Chapter 39**

36 CONFIG.SYS file under DOS 3.3

Because this is a book on batch file programming, the title for this chapter immediately suggests a major question, why am I covering the CONFIG.SYS file?

The answer is fairly simple. The batch file you use the most is your AUTOEXEC.BAT file, and you will end up spending a good bit of time fine-tuning your AUTOEXEC.BAT file to get the best performance out of your system. Because the AUTOEXEC.BAT file works hand-in-hand with the CONFIG.SYS file, it's important that you be able to fine-tune both of them to get the exact performance you want out of your system. Plus, the CONFIG.SYS file is structured very much like a batch file so it makes good sense for this book to cover it.

The CONFIG.SYS commands in DOS 3.3 are not innovative or fancy. Rather, they represent a solid foundation. Most of these commands work under all versions of DOS. They also represent the majority of the commands actually used to configure your system. Because many of you will have some contact with different versions of the operating system, I have elected to separate out this common pool of knowledge into its own chapter. Once we have covered this basic information, we will spend the next two chapters covering commands specific to the two recent versions of DOS.

Getting started

The operating system is made up of three components:

- **The Basic Input Output System or BIOS.** This is made up of two parts: a hardware chip that comes with the computer, and a software portion that's read from the disk.
- **The DOS kernel.** This is the software portion of the operating system that provides the interface between your application programs and the hardware.

- **COMMAND.COM.** This is the command interpreter. It reads the commands you enter into the computer and takes appropriate action.

When you first turn on your computer, the BIOS goes through several steps to get the computer ready to use. The first thing it does is a power-on self test or POST. The POST checks the reliability of RAM (some clones skip this step—others let you skip it by pressing the Esc key) and initializes certain chips and disk drives and other standard equipment.

Problems booting

If something fails the POST test, the computer might freeze. The computer might also display a cryptic error message. A failure at this point might be something serious like a bad memory chip or something as simple as the keyboard connector being loose. Newer computers with setup information stored in a CMOS memory chip will fail at this point if the contents of that chip have been lost or don't agree with your current configuration. If your computer fails at this point, you can try a couple of things:

- If you've added memory, changed boards, or otherwise changed the configuration of your computer, boot off the setup disk or press the appropriate keystroke combination during booting to load the internal setup program and reconfigure the computer.
- If you haven't changed the configuration, you should still try booting off the setup disk. Your battery could have failed and the computer would have lost all setup information. Generally, if running the setup program will correct the problem, the error message will suggest running the setup program.
- If the problem isn't with the setup, try pushing the F1 key several times. An old Compaq I used to use always gave me an error message when it booted, and pressing F1 always cleared it. I was never able to identify the reason, but it's never caused any other problems so I just lived with the annoyance until I replaced the computer.
- Try turning the computer off for thirty seconds or so and then turning it on again. Sometimes, random problems will crop up and turning the computer off for thirty seconds gives all the internal memory a chance to lose its contents.

If all of these things fail, you'll need to either get your computer serviced or use the error message number to track down the problem yourself.

After the POST, the boot record is loaded into memory and executed. It doesn't know very much. In fact, all it knows is how to start DOS. This program is called a *bootstrap program*. The bootstrap routine isn't very smart; it could just as easily be loading different operating systems. All it does is read the beginning of a disk. If there's a disk in the A drive, it will try to read DOS from that disk. If the A drive is empty, it will try to read DOS from the first hard disk—usually the C drive. If DOS isn't present or if the disk is defective, you'll get a non-system disk error message.

You might be wondering why the operating system isn't built into the computer. It is, in fact, built into some Tandy computers.

The problem with building the operating system into hardware is that it makes upgrading to a newer operating system expensive and difficult. Over the past few years, there have been so many improvements to DOS that I wouldn't want to still be locked into an older system simply because it was built into my machine.

Additionally, having a built-in operating system would also prohibit you from running an alternative operating system.

Components of DOS

DOS is made up of three primary programs: IBMBIO.COM (IO.SYS under DOS 6), IBMDOS.COM (MSDOS.SYS under DOS 6), and COMMAND.COM—although the names of the first two files might be different on your system. IBMBIO.COM and IBMDOS.COM are hidden files; you don't see them when you perform a directory. IBMBIO.COM is a supplement to the internal ROM BIOS routines already in the computer. IBMDOS.COM handles input-output operations. COMMAND.COM is the command interpreter.

The bootstrap program in the boot sector loads IBMBIO.COM and IBMDOS.COM. Control is passed to IBMBIO.COM, which initializes all the hardware that's attached to the computer. That's why all the disk drive lights flash and the print head on your dot matrix printer moves to the left. Hardware initialization also moves the part of the operating system from its load position in memory to its final position in low memory and erases the IBMDOS.COM routines in memory that are no longer needed.

After initializing the hardware, IBMBIO.COM looks for and processes a file called CONFIG.SYS. This is short for CONFIGure the SYStem. If no CONFIG.SYS file is found, IBMBIO.COM continues with the boot process. Like the AUTOEXEC.BAT file, the CONFIG.SYS file must be in the root directory of the boot disk. Unlike the AUTOEXEC.BAT file, however, the CONFIG.SYS file contains unique commands. Most of these commands can't be used outside the CONFIG.SYS file. While these commands are unique, they're very similar to batch commands in their use and structure.

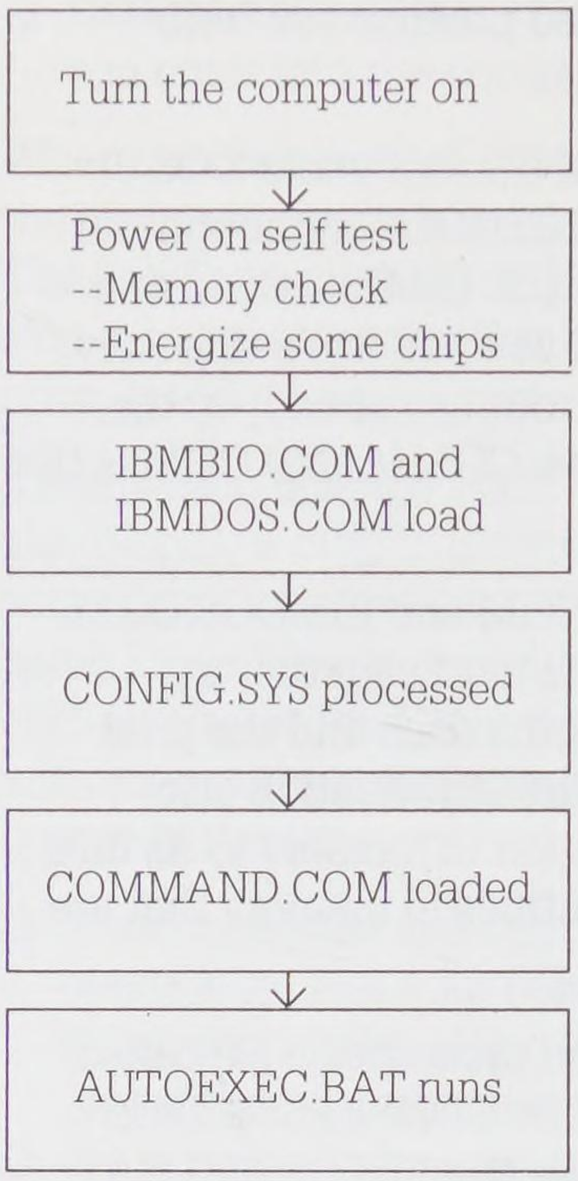
After the CONFIG.SYS file is found and processed, control is passed to IBMDOS.COM. This program loads routines that control the information flow to and from the computer and other hardware, like disk drives and printers. After IBMDOS.COM is finished, COMMAND.COM is loaded. When COMMAND.COM first begins operation, the last thing it does is look for a special file called the AUTOEXEC.BAT file, and it looks for this AUTOEXEC.BAT file only in the root directory of the boot disk. If it finds the AUTOEXEC.BAT file, it executes it just like any other batch file. This makes the AUTOEXEC.BAT file an excellent place to perform tasks that are performed only once, like setting the date and time and loading memory-resident software. This is explored in detail in Chapter 39.

Figure 36-1 illustrates the steps that take place when you switch your computer on. Figure 36-2 shows how the computer's memory is configured with the computer running a program.

The CONFIG.SYS file

The CONFIG.SYS file is an ASCII file that's very much like a batch file, with commands that control how much memory DOS allocates to certain tasks, and how it performs several other tasks. DOS uses it to control the configuration of the computer. In order to add a mouse or connect to a network, you enter commands into the CONFIG.SYS file. If there isn't a CONFIG.SYS file in the root directory, DOS skips this step and loads COMMAND.COM.

The CONFIG.SYS file is structured just like a batch file and must be an ASCII file. It must have one command on each line. Unlike a batch file, however, most commands in the CONFIG.SYS file are unique to the CONFIG.SYS file and can't be



36-1
What your computer does when you switch on the power.

BIOS area
Device drivers loaded from CONFIG.SYS file
PSP of COMMAND.COM (program segment prefix)
Low-memory portion of COMMAND.COM
Master environment block (This is the copy of the environment accessed and modified by batch files.)
Memory-resident software (Along with a full copy of the environment for each program.)
Current program's copy of the environment
DOS memory control block (MCB) for the current program
Current program's PSP
Current program's code area
Current program's data area
Free memory

36-2
Configuration of a computer's memory when it's running a program, including space for memory resident software.

used elsewhere. There's a reason for this. The CONFIG.SYS file runs before COMMAND.COM loads and configures certain aspects of memory usage. Once COMMAND.COM is loaded, there's no room to change the values set by the CONFIG.SYS file. Because the CONFIG.SYS file isn't a batch file, it can't contain batch file commands.

Nine commands can be used in a CONFIG.SYS file under DOS 3.3, and most of the commands can be used under earlier versions of DOS. For the most part, they can be in any order. Keep in mind that, while there are nine commands you can use in your CONFIG.SYS file, only three are used a great deal: BUFFERS=, FILES=, and DEVICE=. The list of CONFIG.SYS commands is described next.

BREAK=ON/OFF

Normally, DOS checks for a Ctrl-Break only during standard input, output, when printing, or when accessing the communications port. This is the default setting of `BREAK OFF`. Setting `BREAK ON` causes DOS to check for a Ctrl-Break whenever a program requests any DOS function. Of course, because DOS has to spend time checking for a Ctrl-Break, DOS will run a little slower with BREAK turned on. The OFF setting is redundant because it's the default.

I recommend the ON setting. If a batch file you're working on hangs when DOS is performing some action where it's not checking the keyboard, then you're going to have to reboot to regain control.

The BREAK command is one of the few CONFIG.SYS commands that you can enter from the keyboard or in a batch file. The syntax is identical to the syntax in a CONFIG.SYS file. Entering `BREAK` alone (from the keyboard or batch file but not in the CONFIG.SYS file) will display the status of Break.

BUFFERS=n

DOS sets up a special place in memory, called a *buffer*, to hold information that's just been read from disk. If the program requests the same information again, DOS can supply it from very fast memory (the buffer) rather than the very slow disk—a process called *disk caching*. Up to a point, the larger the buffer, the faster the disk access system works. And the larger the buffer, the more memory it takes and the less you have available for other uses.

In addition to saving time on a disk read, the buffer saves time on a disk write because, before DOS can write a record to a sector, it must first read that sector. So a large buffer saves on both read and write time. The number *n* is between 1 and 99. The default value that DOS uses if there's no BUFFERS command between 2 and 15, this depending on both the version of DOS you have and your machine's configuration. Each buffer causes DOS to allocate 512 bytes to the buffer. This exactly matches the 512-byte disk sectors that DOS uses. Due to internal overhead, each buffer reduces the amount of memory available to other applications by 528 bytes. So a BUFFERS value of 10 causes DOS to allocate 5280 bytes, or a little over 5K, to the buffer.

A higher buffers statement can dramatically speed up some operations. A program that reads the same information from disk over and over, like a database program, gains a great deal from either a higher level of buffers or a caching program. A program like Lotus, which reads its worksheet into memory and holds the entire set of data in memory, gains little or nothing. In order to load a file into memory, DOS must first load the proper directory. Therefore, systems with a large number of files in a directory or a complex subdirectory structure also gain from a higher buffer setting.

With all the memory-resident programs around, memory is already tight. Making your buffer too large might even slow down a program because it reduces the memory available to the program. Many commercial programs want you to set your buffer to 20. So unless you want to spend a lot of time experimenting to find the best setting or if you have a caching program, just set it to 20 and forget it. In fact, making your BUFFERS setting too high with some versions of DOS can dramatically slow DOS down, as opposed to having no buffers at all.

The DOS BUFFERS command is a very simple cache program. If you install a commercial caching program, like Super PC-Kwik, the installation instructions will probably tell you to set the buffer statement to a very low number—typically one or two. This avoids having two caching programs trying to cache your hard disk. You must have a statement in your CONFIG.SYS file setting BUFFERS to this value. Otherwise, it can default to a value as high as 15.

You might be wondering why you would want a separate caching program, because the BUFFERS setting is really a cache. The problem is that BUFFERS is a very simple cache. Before checking the hard disk for data, DOS first checks the buffers. Its search method is very simple; it starts with the first buffer and checks every single buffer sequentially until it either finds the information it's looking for or it checks all the buffers. If DOS is always reading new information from the disk but first has to sequentially check all the buffers, the buffers can actually slow down the computer! That's why the DOS manual warns about setting buffers too high.

On the other hand, a good caching program doesn't sequentially check its entire contents. In fact, most caching programs allow you to create multi-megabyte caches, way too large to check sequentially. Rather, they generate an index as data is being placed in the cache. All they do when DOS requests information is briefly check their index. That tells them right away if the cache has the data DOS is requesting and, if so, where to get it.

COUNTRY=n

COUNTRY=*n* controls how the date and time are displayed. It also affects the information DOS supplies to applications regarding currency symbols, decimal separator, sorting order and more. The n represents the country code. TABLE 36-1 shows all the COUNTRY codes, along with the corresponding default KEYB value. The default is USA if no value is entered, so English-speaking Americans can ignore it.

DEVICE=

A number of devices are attached to your computer, including disk drives, a hard disk, and printers. Some of these devices are unique and require a device driver in order to communicate with your computer. A *device driver* is a special program provided by the manufacturer to allow a special device to communicate with DOS. A device driver usually ends with the extension .SYS, although .EXE and .COM are also possible extensions. By using device drivers to control special types of equipment, DOS is device-independent. You can use almost any type of equipment with DOS as long as a device driver has been written for the equipment.

Device drivers are attached to DOS in the CONFIG.SYS file. For example, a sample CONFIG.SYS file is shown here:

```
FCBS=1,1
DEVICE=C:\WINDOWS\HIMEM.SYS
```

```
DEVICE=C:\CDROM\SONY_CDU.SYS/D:MSCD210/U:1/B:340/M:H/Q:*/T:*
DEVICE=C:\MOUSE\MOUSE.SYS /Y
DEVICE=C:\SYSLIB\ANSI.SYS
DEVICE=C:\HARDCARD\ATDOSXL.SYS
DEVICE=C:\WINDOWS\SMARTDRV.EXE /DOUBLE_BUFFER
SHELL=C:\SYSLIB\COMMAND.COM C:\SYSLIB\ /E:800 /P
FILES=50
BUFFERS=20
BREAK=ON
LASTDRIVE=E
```

Table 36-1
Possible values for the
COUNTRY configuration command.

Country	Code	KEYBxx
United States	001	US
Canada (French)	002	CF
Latin America	003	LA
Netherlands	031	NL
Belgium	032	BE
France	033	FR
Spain	034	SP
Italy	039	IT
Switzerland	041	SF, SG
United Kingdom	044	UK
Denmark	045	DK
Sweden	046	SV
Norway	047	NO
Germany	049	GR
Australia (English)	061	----
Japan	081	----
Korea	082	----
People's Republic of China	086	----
Taiwan	088	----
Asia (English)	099	----
Portugal	351	PO
Finland	358	SU
Arabic	785	----
Hebrew	972	----

Note that the path isn't set when DOS reads the CONFIG.SYS file. Therefore, the device driver must be in the root directory or the command must supply the complete path to the driver, as my CONFIG.SYS does.

When DOS attaches a device driver, that driver becomes a part of DOS. As you can imagine, writing a device driver is complex and difficult. Most memory-resident software would function better if it were written as a device driver. However, the

task of writing device drivers as complex as most memory-resident software is too difficult for even big software companies.

There are a couple of device drivers of particular note. They are listed as follows:

- ANSI.SYS. ANSI.SYS was first included with DOS 2.0 and has been standard ever since. ANSI.SYS replaces the screen and keyboard handling of DOS with more powerful routines. See Chapter 21 for more information on ANSI.
- VDISK.SYS. VDISK.SYS allows you to set aside part of your memory as an electronic floppy disk drive, called a RAM disk. The advantage of a RAM disk drive is speed: they're extremely fast. The disadvantage of a RAM disk is that all the data in them is lost when you turn the computer off or if you lock up the computer, and a RAM disk reduces the memory available to other applications.
- DISPLAY.SYS, DRIVER.SYS, and PRINTER.SYS. DRIVER.SYS lets you define an external drive as a logical drive. The primary use for DRIVER.SYS is to define an external 5.25" drive as a logical drive on a machine with only 3.5" drives. The other two drivers, DISPLAY.SYS and PRINTER.SYS, are used for displaying and printing character sets from other countries.

FCBS=

This specifies the number of file control blocks (FCBs) that DOS can concurrently open. The format is

```
FCBS=m,n
```

where the *m* gives the total number of files that can be opened simultaneously by FCBs. The default is 4 and the range of possible values is 1–255. The *n* specifies the number of protected file openings—that is, the first *n* FCBs are protected against being closed if a program tries to open more than m files using FCBs. The default is 0 and the range of possible values is m–255.

DOS 5 only supports specifying one number for the FCBS command in the form

```
FCBS=m
```

where the *m* has the save meaning as it does above.

Most programs don't use FCBS anymore, but you can save some memory by adding the `FCBS=1` command to your CONFIG.SYS file because the default is 4 if you do not have an FCBS statement. If one of your applications complains about being unable to open files, then either increase the number used with FCBS in your CONFIG.SYS or upgrade your application to a more modern version.

FILES=#

FILES replaced FCBS as the way to open files. The `FILES=#` command controls how many files the entire system can have open at any one time. The default value is 8, which is too few for many applications. The maximum number of files that a process can have open is 20. This twenty includes five predefined files: input, output, error, auxiliary, and printer files. Several major programs, like dBASE, require a FILES value of 20. Some programs offer the ability to shell to other programs, for example, /System in Lotus, Run a program in Wordstar, Shell in Basic, or Escape transfer run in Microsoft Word. The second application is called a child process. The shelling process can open twenty files, as can the second application, so it's possible (although unlikely) that you could require a FILES specification as high as 40. A very few programs require a value as high as 99.

If you don't specify a value, the default is 8. Each file above 8 requires about 60 bytes. The maximum value is 99. As a general rule, set your FILES value to 40. If you ever need more, you'll get an error message like "No free file handles." If that happens, you need to increase the value of the files statement in your CONFIG.SYS file and reboot. Of course, each file requires only a small amount of memory, so setting it too high won't affect your memory too much.

LASTDRIVE=n

This command controls how many disk drives DOS appears to have. The *n* represents the last drive in the system, which, on a typical XT, would be C. This number only sets an upper limit; DOS still tracks the correct number. Note there isn't a colon after the drive letter. Remember that LASTDRIVE must leave room for any non-disk drives you have. These include RAM disks, any unusual disk or tape drives added as device drivers, any hard disks that have been partitioned into more than one drive, and any "pseudo" drives set up with the SUBST command. Each drive letter reserved beyond the C drive uses 88 bytes of memory.

If not specified, LASTDRIVE defaults to the E drive. That means if you only have a C drive on your system, you can save memory by adding a `LASTDRIVE=C` command to your CONFIG.SYS to prevent DOS from allocating memory for the D and E drive you do not have.

Certain software can overwrite the LASTDRIVE setting. Typically, network software will define several drive letters beyond the LASTDRIVE setting as network drives. For example, on a network computer I used to use, I had a `LASTDRIVE=D` setting. Once the Novell network software loads, however, it defines P, Q, R, S, Y, and Z as network drives.

SHELL=

The DOS command interpreter is called COMMAND.COM. You don't have to use COMMAND.COM; you can use any other one, such as the NDOS program that comes with the Norton Utilities of the shareware 4DOS program. You specify another command interpreter with the SHELL command in the CONFIG.SYS file. If, like device drivers, the command interpreter you want to use isn't in the root directory, you must specify the full path. The SHELL command is also used to expand the DOS environment.

The SHELL command can't be used at the DOS prompt. In addition, COMMAND.COM doesn't show up in the environment under SHELL but under COMSPEC. You can change the location where DOS looks for COMMAND.COM at the DOS prompt with the COMSPEC (or optionally SET COMSPEC) command, like so:

```
SET COMSPEC=C:\COMMAND.COM
```

or

```
COMSPEC=C:\COMMAND.COM
```

In addition to pointing to COMMAND.COM, the SHELL command is used to expand the size of the DOS environment. This was explained back in Chapter 13.

STACKS=

STACKS controls the number and size of the stack frames. The stack frame is an area of memory used by the interrupt handler to store the information it needs to

restore the state of the computer when it finishes processing an interrupt. The command is

STACKS=*m*,*s*

where *m* is the number of stack frames. The possible values are 0 or 8–64, and the default is 0. The *m* value increases memory usage by the size specified by *s*. The *s* value specifies the size of each stack frame. The possible values are 0 or 32–512. The default on an 8088 is "0,0" and on all other systems the default is "9,128."

STACKS are designed to protect faster machines from too many hardware interrupts occurring close together. Most computers run fine without this protection, so adding the command STACKS=0,0 to your CONFIG.SYS file will free up about 1K of memory. If you should receive the "Internal stack overflow, system halted" error message, then increase the value for STACKS.

Summary

The operating system is made up of three components: the basic input output system or BIOS, the DOS kernel and COMMAND.COM.

When a computer begins to boot, it goes through a power-on self-test or POST to check the hardware for problems.

If the computer encounters problems during the POST, it might not boot or might require you to run the setup program.

After running the POST, the bootstrap program loads the boot record into memory and executes it.

DOS is made up of three primary programs: IBMBIO.COM, IBMDOS.COM and COMMAND.COM.

The bootstrap program loads IBMBIO.COM and then IBMDOS.COM.

IBMBIO.COM loads and processes the CONFIG.SYS file.

The BREAK command in the CONFIG.SYS file controls how often DOS checks to see if you have pressed Ctrl-Break.

The BUFFERS command in the CONFIG.SYS file controls how much memory DOS allocates for disk caching.

The COUNTRY command in the CONFIG.SYS file controls what format DOS uses for dates, time, currency symbols, decimal separators, sorting order and more.

The DEVICE command in the CONFIG.SYS file allows DOS to load device drivers to allow it to communicate with additional devices.

The FCBS command in the CONFIG.SYS file controls how many files DOS can open using the old MS DOS 1.x data structure.

The FILES command in the CONFIG.SYS file controls how many files DOS can have open at once, including predefined files like input, output, error, auxiliary and printer.

The LASTDRIVE command in the CONFIG.SYS file specifies the highest drive letter DOS is to allow.

The STACKS command in the CONFIG.SYS file controls the number and size of the stack frame.

After it finishes, IBMBIO.COM loads COMMAND.COM.

COMMAND.COM loads and processes the AUTOEXEC.BAT file.

37 CONFIG.SYS file under DOS 5

DOS 5 did not add any major new functions to the CONFIG.SYS file but it did add a wealth of memory management features. This chapter will look at the features new in DOS 5. "Standard" CONFIG.SYS commands were covered in the last chapter.

BUFFERS

While BUFFERS were included in earlier versions of DOS and were covered in the last chapter, they were improved in DOS 5. If you're running DOS 5 or 6 on a 286 or better machine, then adding the `DOS=HIGH` command to your CONFIG.SYS file causes disk buffers defined with the `BUFFERS=`*n* command to be loaded into high memory if *n* is 48 or less. This reduces BUFFERS allocation of conventional memory to 512 bytes.

You can even speed things up more by using the disk caching program included with DOS 5, SmartDrive. First change your BUFFERS to a low value with a command like `BUFFERS=4`. BUFFERS uses an unsophisticated caching algorithm and does not do as good a job as SmartDrive. Setting its value to four keeps it out of the way.

Next, add the command `DEVICE=C:\DOS\SMARTDRV.SYS 1024` to your CONFIG.SYS file. Of course, you need to specify the path to SMARTDRV.SYS on your system. This command causes SmartDrive to use 1M of extended memory for a cache. After you reboot, you should notice faster disk accesses.

COMMENT=

The COMMENT= command is an "undocumented" CONFIG.SYS file command. That is, Microsoft does not tell you about it and does not promise to support it in

future releases. Nevertheless, it is supported in the DOS 6 beta version I'm currently using.

The command `COMMENT=Letter` causes DOS to ignore any line beginning with that letter. For example, the command `COMMENT=B` would cause DOS to ignore the BUFFERS line in your CONFIG.SYS file if it came after the COMMENT= line.

The only real use for the COMMENT= command is to "fake out" installation programs that want to read and modify your CONFIG.SYS file. For example, if you use a disk caching program, you might have a very low value for BUFFERS, but some installation programs will try and change it. You can fake them out with a series of commands like this:

```
BUFFERS=8
COMMENT=B
BUFFERS=40
```

Because of the additional complexities introduced to the CONFIG.SYS file in DOS 6, it is unlikely that installation programs will continue to make massive alterations to your CONFIG.SYS file, so this command is of limited use. It can also make it very hard for someone else to read and understand your CONFIG.SYS file.

DEVICEHIGH

DEVICEHIGH is the same as the DEVICE command, except that it tries to load the device driver into the high memory created by EMM386.EXE rather than conventional memory. If this fails, it goes ahead and loads the device driver into conventional memory. The two device drivers EMM386.EXE and HIMEM.SYS cannot be loaded into high memory, so you should use the DEVICE command with them.

DOS

This command specifies where DOS should be loaded. The command `DOS=HIGH` loads it into high memory, and the command `DOS=UMB` causes it to allocate upper memory blocks [UMB] created by EMM386.EXE for its own use in loading memory resident software.

You can combine these two statements into `DOS=HIGH,UMB` to load DOS itself into high memory and to have it allocate space for loading memory resident software into high memory. Doing this requires loading the HIMEM.SYS and EMM386.EXE device drivers.

DRIVPARM

The DRIVPARM command is used to modify how floppy disk drives think of themselves. For example, if you install a 3.5" drive on an older computer, it might think of itself as a 360K drive. Adding the command `DRIVPARM=/D:1 /F:2 /I` to your CONFIG.SYS file will correct the problem. Most newer computers do not need this command.

INSTALL

INSTALL loads a memory resident program from the CONFIG.SYS file rather than the AUTOEXEC.BAT file. This causes the program to use a little less memory, but the process is tricky. You're best to avoid INSTALL unless a program specifically states that it is compatible with the INSTALL command.

REM

The REM command was added in DOS 4 to give the CONFIG.SYS file support for documentation remarks. Anything following a "REM " command is ignored. This can be especially useful when experimenting with your CONFIG.SYS file because you can "turn off" any command without having to remove it by adding a REM command to the front of the command to turn off.

SWITCHES

The SWITCHES command is used to cause DOS to treat an enhanced 101-key keyboard just like a conventional keyboard. It also tells DOS that WINA20.386 has been moved out of the root directory—although this requires altering your Windows' SYSTEM.INI file as well.

Summary

If you load DOS high and you have less than 48 BUFFERS, then DOS will load your BUFFERS into high memory.

The COMMENT= command can be used to fake out installation programs by adding commands to your CONFIG.SYS file that DOS ignores.

The DEVICEHIGH command performs the same function as the DEVICE command but it loads the device driver into high memory (if it will fit) rather than conventional memory.

The DOS command specifies where DOS should be loaded (high or conventional memory) and if upper memory blocks should be reserved for its use.

The DRIVPARM command is used to supply specifications to floppy disk drives when those drives do not configure themselves properly.

The INSTALL command is used to load memory resident programs in the CONFIG.SYS file rather than the AUTOEXEC.BAT file. Not all memory resident programs support this and it should be used with extreme care.

The REM command is used to add comments to the CONFIG.SYS file and to temporarily "comment out" commands so they do not execute.

The SWITCHES command controls how DOS handles enhanced keyboards and, in conjunction with SYSTEM.INI, controls where Windows looks for the file WINA20.386.

38 CONFIG.SYS file under DOS 6

While DOS 5 only added a few memory configuration commands to the CONFIG.SYS file, DOS 6 added an entire programming language! This programming language is dedicated primarily to menu construction and lets you construct some very powerful menus. These menus let you boot your computer with any one of as many different configurations as you like. In this chapter, we will explore these powerful extensions. If you have not done so already, you should go back and read Chapters 36 and 37 first.

Show us your numbers

Probably the single most common hint in the computer magazines are the short programs you can add to your AUTOEXEC.BAT file to turn off NumLocks. It seems that almost no one wants the computer to boot with NumLock on. Well, no more. Just add the command `NUMLOCK=OFF` to your CONFIG.SYS file and the computer will now boot with NumLock turned off. If you like, you can build the command into the menus covered in this chapter so that some configurations load with NumLock on and others load with it off.

To load or not to load—that is the question

I have an external mouse for my notebook computer, so I put a `DeviceHigh=MOUSE.SYS` command in its CONFIG.SYS file to load the mouse driver. However, when I take the notebook out for a quick session, I don't always take the time to attach the mouse—especially if I'll be running a DOS program. The computer does not know I decided to skip the mouse so MOUSE.SYS takes several seconds trying to find the mouse before it gives up. What I need is a way to easily tell the computer to skip loading the mouse but just this time.

The question mark does just that. If you precede the equal sign for any DEVICE or DeviceHigh command with a question mark, DOS will stop on that line and ask you

if you want to load that driver. So, if I change the mentioned line to `DeviceHigh?=MOUSE.SYS`, then each time I boot DOS will stop on that line and ask me the "DeviceHigh=MOUSE.SYS [Y/N]?" question and wait for my answer before proceeding.

Not only can this save you time, as it does in the mouse example, but it can also save you memory. My desktop computer has a CD ROM drive attached. That CD ROM requires a device driver that takes up a significant amount of memory. By adding a question mark to that line, I can bypass loading the CD ROM driver when I know I'm going to need a lot of memory for something else.

As we'll see later, menus can handle this job even better. In fact, with intelligently constructed menus, you will rarely have a use for this option.

How do I boot thee? Let me count the ways

It's a rare computer user who never needs to change her boot configuration. As I mentioned earlier, my desktop system has a CD ROM drive that I sometimes want to skip when I'm running a program that needs a lot of memory. At the same time, it would be nice if I could skip loading ANSI.SYS and use a smaller environment to save even more memory.

At my last job, we had a network that loaded with two device drivers that took up a lot of memory. Because some of my files were on network drivers and the network handled all printing, I usually booted with the network. However, I sometimes worked with a huge Lotus spreadsheet that needed every byte of memory I could squeeze out of my computer. At other times, I wanted to use my single user copy of Ventura Publisher—which refuses to load if it detects a network! In this situation, I needed three different configurations:

- my usual configuration
- a maximum memory configuration
- a no-network configuration

Prior to DOS 6, all you could do was keep different copies of your AUTOEXEC.BAT and CONFIG.SYS files. When you wanted to reboot with a different configuration, you would copy the appropriate configuration files to the root directory of your C drive and reboot.

Well, no more!

DOS 6 adds a menu language to the CONFIG.SYS file that lets you choose the configuration you want when you boot. Not only that, the menu automatically creates an environmental variable to tell your AUTOEXEC.BAT file which configuration was selected.

Let's look at the process of constructing a CONFIG.SYS file.

Oops!

Prior to DOS 6, working on your CONFIG.SYS file was dangerous. Some mistakes would cause DOS to be unable to process your CONFIG.SYS file; you couldn't break out before getting to the AUTOEXEC.BAT, so the computer would hang. Then you would have to dig out a bootable disk and boot off of it to correct the problem. While it's still a good idea to keep a bootable disk handy, DOS 6 makes it much easier and safer to work on your CONFIG.SYS file.

If there is a mistake in your CONFIG.SYS file and you want to bypass it, just press and hold either Shift key or press the F5 key when you see the "Starting MS-DOS"

message on the screen. DOS will bypass both your CONFIG.SYS and AUTOEXEC.BAT files and boot with a minimal system. However, if you've used DBLSPACE to compress your drive, that will be loaded. Nevertheless, you will notice some differences:

- If COMMAND.COM is not in the root directory, DOS might not be able to find it. Prior versions of DOS would lock up if they could not find COMMAND.COM, but DOS 6 will simply ask you to enter the path to COMMAND.COM.
- Some drives might not work. If your Hardcard or Bernoulli drives require a drive driver, they will not work. Your mouse will not work, and you might not have extended or expanded memory.
- No environmental variables will be set. You will have the default "C>" prompt and there will be no path, so you will have to either change to the appropriate subdirectory before running a program or enter the full path to that program.

If only one line is causing you problems, then all of these things might be too much to put up with. If you press the F8 key, rather than the F5 key, when you see the "Starting MS-DOS" message, DOS will ask you about each line in your CONFIG.SYS file and about running your AUTOEXEC.BAT file. That way, you can simply skip the offending line.

That's commonly done

The first thing you need to know how to do is to define a block of commands. The first thing you do is pick out a name for that block. It can be as long as you like but cannot contain any spaces. While longer names are allowed, you'll want to keep them fairly short so you can work with them more easily. Also, it cannot be the name "common" (because "common" is a system name with special meaning) or "menu" (because it's used to define a menu). Capitalization does not matter. The name must be left-justified and enclosed in square brackets. So some sample legal names would be [NoNet] or [NetWork].

Now, it's time to take a look at your CONFIG.SYS file. Some of the commands, like `BREAK=ON` and `FILES=40`, are commands you will want every configuration you define to process. Group these together and start them with a common block name, like this:

```
[COMMON]
BREAK=ON
FILES=40
BUFFERS=20
NUMLOCK=OFF
```

The commands that follow a group name belong to that name until another group name is encounterd so, to avoid any confusion, go ahead and group these together at the bottom of your CONFIG.SYS file. Once we're finished, you can move them anywhere you like as long as you keep them together.

If you do move your common block away from the bottom of your CONFIG.SYS file, you should insert an empty common block at the bottom of your CONFIG.SYS file. Some installation programs append commands to the end of your CONFIG.SYS file. If some block other than a common block is defined at the bottom of your CONFIG.SYS file, then the new commands will only be run when you select that particular configuration. However, if you end your CONFIG.SYS file with a common block, commands appended to the end of your CONFIG.SYS file will be run no

matter which configuration you select. A CONFIG.SYS file can have as many different common blocks as you like.

Defining the menu

Our next step is to define the menu options. To begin the process, we start with a `[MENU]` line. This tells DOS that what follows is a menu definition. Because we don't want a blank white-on-black menu, our next line will define the colors for the menu. The format of the command is `MenuColor=`*f*`,`*b* where *f* is the foreground color and *b* is the background color. TABLE 38-1 lists the allowable colors. The command `MenuColor=15,1` will give you my favorite—bright white letters on a blue background.

Table 38-1
The color codes that can be used with the MENUCOLOR command in the CONFIG.SYS file.

Number	Color	Number	Color
0	Black	8	Gray
1	Blue	9	Bright Blue
2	Green	10	Bright Green
3	Cyan	11	Bright Cyan
4	Red	12	Bright Red
5	Magenta	13	Bright Magenta
6	Brown	14	Yellow
7	White	15	Bright White

Next, we need to define the menu items that will appear in the menu. The simplest form for the definition is `MenuItem=`*First* for the first menu option, `MenuItem=`*Second* for the second, and so on. The *First* and *Second* would be replaced with the names of blocks that are defined in the CONFIG.SYS file. A very simple CONFIG.SYS menu might look like this:

```
[MENU]
MenuColor=15,1
MenuItem=Network
MenuItem=NoNet

[NoNet]

[Network]
DEVICE=C:\NETWORK\NETWORK.SYS

[COMMON]
BREAK=ON
FILES=40
BUFFERS=20
NUMLOCK=OFF
```

In this simple example, the `[NoNet]` has no commands under it so all it processes are the "Common" items, while the `[Network]` option has a single device driver under it. The Network device driver is loaded only when the Network option is selected. This would give you a simple menu on the screen that looks like this:

MS-DOS 6 Startup Menu

1. Network
2. NoNet

Enter a choice: 1

What does that mean?

If you boot with this sample CONFIG.SYS file, the opening menu is going to give you two options: "Network" and "NoNet." While you might know what they mean, other users might not. While the naming conventions for the menu blocks are fairly restrictive, you can still use very descriptive names in the menu. If you follow the name of the block with a comma and then a description, that description is used in place of the block name in the CONFIG.SYS menu. So, we might update the sample menu to look like this:

```
[MENU]
MenuColor=15,1
MenuItem=Network,Load The Menu Driver While Booting
MenuItem=NoNet,Skip Loading The Network To Save Memory
[NoNet]
[Network]
DEVICE=C:\NETWORK\NETWORK.SYS
[COMMON]
BREAK=ON
FILES=40
BUFFERS=20
NUMLOCK=OFF
```

Now, when she boots, the user will get two options with very understandable descriptions that look like this:

MS-DOS 6 Startup Menu

1. Load The Menu Driver While Booting
2. Skip Loading The Network To Save Memory

Enter a choice: 1

How long to wait

This menu will wait forever for the user to pick one of the two options. If you change configurations a lot, that's ok but it will be inefficient if you usually boot from a particular configuration. You can define one of the configurations as the default configuration and tell DOS how many seconds (1–90) to wait on the user. If the user does not make a selection within the allocated time, DOS boots using the default configuration. The command is `MenuDefault=`*`Block`*`,`*`Time`* where *`Block`* is the name of the block to use as the default and *`Time`* is the number of seconds to wait.

In the previous example, we could make the network configuration the default with the following:

```
[MENU]
MenuColor=15,1
MenuDefault=Network,20
MenuItem=Network,Load The Menu Driver While Booting
MenuItem=NoNet,Skip Loading The Network To Save Memory
[NoNet]
[Network]
DEVICE=C:\NETWORK\NETWORK.SYS
[COMMON]
BREAK=ON
FILES=40
BUFFERS=20
NUMLOCK=OFF
```

If the user does not make a selection within twenty seconds, the computer boots automatically in the network configuration. While waiting, DOS displays a clock on the screen showing the number of seconds remaining to make a selection.

Communicating with AUTOEXEC.BAT

Some of the menu options you might select in your CONFIG.SYS file will have implications for your AUTOEXEC.BAT file. For example, when I skip loading the CD ROM device driver on my desktop system, I also want to skip loading the CD ROM control program in my AUTOEXEC.BAT file. DOS 6 gives us two very similar ways of communicating between the CONFIG.SYS and AUTOEXEC.BAT files.

The first way is handled automatically by DOS itself. When you select a particular menu option, the name of that menu option is stored under the environmental variable "CONFIG" using the same capitalization you used in the menu. So, if you select the "NoNet" option from the above menu, then there is an environmental variable `CONFIG=NoNet`. If your network also requires a memory resident program to be loaded in your AUTOEXEC.BAT file, then it could have a line like `IF %CONFIG%==Network NET-TSR` to load the memory resident network driver only when the user selected the Network option.

However, this can become complex if you have a lot of menu items. For example, imagine the situation where you are selecting from six different configurations and you want to load the network under four of them, a CD ROM under three of them, and a fax board program under two of them.

To overcome this complexity, DOS 6 now allows you to define environmental variables in the CONFIG.SYS file. The syntax is exactly the same as when they are defined in the AUTOEXEC.BAT file so the command `SET CDROM=Yes` in the CONFIG.SYS file will create an environmental variable called "CDROM" containing the value "Yes."

For the complex example I mentioned earlier, each menu option could create three environmental variables—NETWORK, CDROM and FAX—and give them a value of "Yes" if the AUTOEXEC.BAT file is to load that program and a value of "No" if it is

to skip that option. That way, the AUTOEXEC.BAT would only need the three following lines to process all these options properly:

```
IF %NETWORK%==Yes NET-TSR
IF %CDROM%==Yes C:\CDROM\MSCDEX /d:MSCD210 /M:4 /L:E /V
IF %FAX%==Yes C:\FAX\FAXSEND /V
```

As you can see, this makes it very easy for the AUTOEXEC.BAT file to "follow in the footsteps" of the CONFIG.SYS file and you end up with exactly the configuration you ask for.

Other stuff

That's pretty much all the information that most of you will need to construct your CONFIG.SYS file in a way that gives you exactly the configuration you want. For most of you, I suggest you skip this "Other Stuff" section. However, a few of you will need to define complex menus that can benefit from the following information.

A menu cannot have more than nine MenuItems in it. If you need more than nine different configurations, then you can define one or more MenuItems as a SubMenu. A SubMenu is nothing more than a nested menu. When you select the SubMenu, it brings up yet another menu. The following example illustrates this:

```
[Menu]
MenuColor=15,1
MenuDefault=Three,20
MenuItem=One,Skip SubMenu
MenuItem=Two,Also Skip SubMenu
SubMenu=Three,Run SubMenu
[One]
[Two]
[Three]
MenuItem=SubOne,First SubMenu Item
MenuItem=SubTwo,Second SubMenu Item
MenuItem=SubThree,Last SubMenu Item
[SubOne]
[SubTwo]
[SubThree]
```

If the user selects either of the first two options ("Skip SubMenu" or "Also Skip SubMenu") from the main menu, then the configuration proceeds normally. This example lacks any commands for these options, but you could fill them in as required. However, if the user selects the third option of "Run SubMenu," then another menu pops up, and it too has three options.

You can use a MenuDefault in SubMenus if you like and it will make the specified option the default if you press return. However, the timeout option of MenuDefault does not work for SubMenus.

Users with complex menu structures may also find the INCLUDE command useful. As you can imagine, if you have a lot of menu options where some of them are supposed to load the network and others aren't, typing in separate network code for each menu option that needs it will mean that you will have a

lot of duplicate code. That will be especially bothersome when it's time to upgrade the network.

To avoid this problem, you can set up one section that handles the network, such as this sample:

```
[Network]
DEVICE=C:\NETWORK\NETWORK.SYS
```

Then, to reference it in each menu option that loads the network, use the command `INCLUDE=Network`. You can set up similar sections for the CD ROM, fax board, and any other device you load in some menu options but not others.

An example

What follows is my CONFIG.SYS file as it exists today. As you can see, I start off with some documentation remarks and then have a common block. After that, I have a menu, which is followed by its three menu blocks. After that, I have a memory block that is included in two of the three menu options. Finally, I close with a blank common block in case an installation program appends commands to the end of my CONFIG.SYS file. For most users, this is as complex as your CONFIG.SYS file needs to get. However, it's nice to know that Microsoft has given us these powerful tools so that we can develop as many different configurations as we need.

```
REM NAME:     CONFIG.SYS
REM VERSION:  5.0
REM DATE:     December 23, 1992
REM REVISION: 5.00 Added DOS 6 Commands
[COMMON]
NUMLOCK=OFF
BUFFERS=20,0
FILES=50
LASTDRIVE=E
FCBS=1,0
STACKS=0,0
BREAK=ON
DEVICE=C:\HARDCARD\ATDOSXL.SYS
[MENU]
MenuColor=15,1
MenuDefault=Normal,20
MenuItem=Minimal, Load Minimal System
MenuItem=No_CD_ROM, Load Everything Except CD ROM
Drivers
MenuItem=Normal, Everything Loaded
[Minimal]
SET LOADCDROM=NO
[No_CD_ROM]
INCLUDE=MEMORY
SET LOADCDROM=NO
[Normal]
INCLUDE=MEMORY
SET LOADCDROM=YES
DEVICE=C:\CDROM\SONY_CDU.SYS/D:MSCD210/U:1/B:340/M:H/Q:*/T:*
```

```
[Memory]
SHELL=C:\SYSLIB\COMMAND.COM C:\SYSLIB\ /E:800 /p
DEVICE=C:\SYSLIB\HIMEM.SYS
DEVICE=C:\SYSLIB\EMM386.EXE RAM X=D100-D1FF X=D100-D8FF
DOS=UMB
DOS=HIGH
DEVICEHIGH C:\MOUSE\MOUSE.SYS /Y
DEVICEHIGH C:\SYSLIB\ANSI.SYS /X
DEVICE=C:\SYSLIB\SMARTDRV.EXE /DOUBLE_BUFFER
[COMMON]
```

Adding the command NUMLOCK=OFF to your CONFIG.SYS file under DOS 6 will cause the computer to boot with NumLock turned off.

Summary

Adding a question mark before the equal sign in a DEVICE= or DeviceHigh= statement will cause DOS to prompt you before loading that device driver.

Pressing the F5 key or either shift key while DOS is booting and displaying the "Starting MS-DOS" message will cause DOS to bypass your CONFIG.SYS and AUTOEXEC.BAT configuration files.

Pressing the F8 key while DOS is booting and displaying the "Starting MS-DOS" message will cause DOS to ask you about each line in your CONFIG.SYS file and about running your AUTOEXEC.BAT file.

Placing a name inside square [brackets] defines that as the beginning of a block in your CONFIG.SYS file.

The block named [Common] is executed no matter what configuration you end up selecting so it's a good spot for common commands. You can have more than one [Common] block if you like.

You need to have a [Common] block at the end of your CONFIG.SYS file—even if it is empty—so commands appended to your CONFIG.SYS file by installation programs will run under any configuration you select.

The command [MENU] marks the beginning of a menu.

The foreground and background colors of the menu can be changed with the MenuColor command.

The command MenuItem=Block defines a menu item. The "Block" portion of the command must correspond to a block starting with the same word, so MenuItem=Mouse requires a corresponding [Mouse] block.

If you follow the name of the block in a MenuItem command with a comma and description, that description is shown in the menu rather than the block name. This description can have spaces in it.

The MenuDefault command can be used to specify the default configuration and to tell the computer how long to wait on the user to make a selection before using the default.

When making a configuration selection using a CONFIG.SYS menu, then DOS automatically creates a "CONFIG" environmental variable containing the name of

the menu item that was selected. The SET command can be used in the CONFIG.SYS to create additional environmental variables to communicate with the AUTOEXEC.BAT file.

The Menu command allows at most nine menu options. To overcome this limitation, the SubMenu command allows you to define a menu within a menu, so selecting this option from the main menu brings up a secondary menu.

A menu option can run another block of commands by using the Include command.

39 AUTOEXEC.BAT, WINSTART.BAT, & shutdown files

The last step the computer performs when it boots is checking for the AUTOEXEC.BAT file in the root directory. If it finds this file, it runs it automatically—hence the name AUTOmatically EXECuted BATch file. Because it runs automatically when you boot or reboot, the AUTOEXEC.BAT file is the perfect place to do those things you only do once, like setting the path and loading memory-resident software. With one very minor exception, the AUTOEXEC.BAT file is just like any other batch file. There are no special AUTOEXEC.BAT commands. The only commands you can use in an AUTOEXEC.BAT file are the same commands you would use in any other batch file.

Two minor differences

One unusual but very minor difference of the AUTOEXEC.BAT is that, if no AUTOEXEC.BAT file exists, the last thing DOS will do is prompt you for the date and time. If an AUTOEXEC.BAT file does exist, DOS will turn control over to it and never prompt for the date and time.

Older computers didn't come with a built-in clock. They required a DATE and TIME command in the AUTOEXEC.BAT file so that the user could set the clock manually. Later, vendors began selling boards containing a clock. These required running a program in the AUTOEXEC.BAT (some used a device driver in the CONFIG.SYS file) to transfer the date and time from the clock to DOS. Almost all newer computers have a clock built in that automatically communicates with DOS, so most users can ignore the clock.

☑ Remember: With an AUTOEXEC.BAT file, DOS does not ask you for the date and time. Without one, it asks you automatically.

The second unusual aspect of the AUTOEXEC.BAT file is the contents of its %0 replaceable parameter when running. Normally, the %0 replaceable parameter contains the name of the batch file exactly as it was entered on the command line. However, when DOS automatically runs AUTOEXEC.BAT while booting, there is no command line for it. As a result, the %0 replaceable parameter (and all the others too) are empty. This is not true if you run the AUTOEXEC.BAT file manually from the command line.

While minor, this has an unusual use. Some of your AUTOEXEC.BAT commands can be executed more than once per session without problems. These include creating the path and any environmental variables you use. Others, like loading memory resident software, should only be executed once per session. When the test `IF (%0)==()` is true, then you know the AUTOEXEC.BAT is running at booting and it should go ahead and run these commands that only run once per session. When this test fails, then the AUTOEXEC.BAT file has been run by the user and should skip over these once-per-session commands.

This knowledge allows you to construct an AUTOEXEC.BAT file that can be run by the user to refresh the path and environmental variables without the danger of reloading memory resident software or performing any other action that should be performed only once per session. This extension is left to the user.

System configuration

As I explained in prior chapters, most system configuration is performed by the CONFIG.SYS file. However, several configuration commands belong in the AUTOEXEC.BAT file. They are as follows.

PATH

As explained in detail in Chapter 13, the path is nothing more than a list of subdirectories for DOS to search when a program isn't in the current subdirectory. The syntax for the PATH command is

```
PATH=C:\;Sub1;Sub2;...;Sub3
```

If your path is

```
PATH=C:\;\DOS;\DATABASE;\WP
```

then DOS will search the following subdirectories in this order:

1. The current subdirectory
2. C:\
3. \DOS
4. \DATABASE
5. \WP

This is normally what you want. However, if you're working on the A Drive, then the path is really

```
PATH=C:\;A:\DOS;A:\DATABASE;A:\WP
```

because A is the default drive. So you're better off to specify the full path, like this:

```
PATH=C:\;C:\DOS;C:\DATABASE;C:\WP
```

A second problem is that the PATH command can contain only the same 127 characters, just like any other DOS commands. Because `PATH=` itself takes five

characters, 122 characters are left over for the path itself. A solution to this problem is the SUBST command. The SUBST command allows you to substitute a drive letter for a subdirectory, so

```
SUBST D: C:\DOS
SUBST E: C:\DATABASE
SUBST F: C:\WP
```

allows you to use D: anywhere you would have used C:\DOS. Your PATH command can now be

```
PATH=C:\;D:\;E:\;F:\
```

This makes the PATH command shorter, as well as easier to read. Generally speaking, you won't have set a path before using the SUBST command. Therefore, SUBST.EXE must be in the root directory, you must change to the directory containing the path before you issue the SUBST command, or you must specify the full path to the SUBST program. Here's an example of having SUBST.EXE in the C:\DOS directory:

```
CD\DOS
SUBST D: C:\DOS
```

or

```
\DOS\SUBST D: C:\DOS
```

Warning: If you use the SUBST command in your AUTOEXEC.BAT file to shorten your path, it is unlikely that you have a path set at the time. Therefore, you must change to the subdirectory containing SUBST.EXE, have a copy of SUBST.EXE in the root directory, or specify the full path to SUBST.EXE.

Keep in mind that if you do not specify a last drive in your CONFIG.SYS file with a LASTDRIVE command, DOS defaults to using the E drive as the last drive. If you plan on using the SUBST command to define several substitute drives, you will need to specify a higher LASTDRIVE value in the CONFIG.SYS file.

If you enter the PATH command with nothing after it, DOS displays the current path. If you enter the path followed by a semicolon, then DOS resets the path to nothing. This causes DOS to search only the default directory for programs and batch files. If you specify a path incorrectly, DOS will not find the error until it needs to search the path. If you enter an invalid directory in the path, DOS ignores that entry.

The PATH command is really an environmental variable. The difference is that you don't have to start the command with a SET command, although you could. Like other environmental variables, the path is accessible to a batch file with its name surrounded by percent signs. A batch file could add the E drive to the path with the command:

```
PATH=%PATH%;E:\
```

This won't work from the command line, only from a batch file.

PROMPT

The default DOS prompt is a "C>," which tells you that C is the default drive. You can use the PROMPT command to make the DOS prompt display a wide range of

information. This is normally done once in the AUTOEXEC.BAT file. The details on using the PROMPT command are shown in Chapter 13.

VERIFY

From the name, you would expect that setting VERIFY to "on" causes DOS to check data it writes to a disk to make sure that it was written properly. You would expect DOS to read back the data as it writes it and compare what it reads back to what it wrote. When they match, it would go on to new data. When they didn't match, it would either retry or signal an error.

This isn't what DOS does, however. When DOS writes information to disk, it also includes a special checksum, called a Cyclical Redundancy Check, or CRC. The CRC is a number calculated mathematically and based on the data on the disk. Writing the same data to disk will always cause DOS to write the same CRC to disk. By including the data and the CRC on the disk, DOS has two versions of the data.

When DOS reads data from disk, it computes the CRC and compares it to the one already read from the disk. If they don't match, DOS knows it's read the data incorrectly. When the two CRCs don't match, DOS tries several more times. If it can't read the data where the CRCs match, it responds with an error message.

When VERIFY is on, DOS does more than write data to disk. After writing the data, it reads the data and computes a new CRC. If the new CRC matches the CRC stored with the data on the disk, DOS assumes the data was written properly. Note that DOS doesn't compare the data on the disk with the data in memory, which would be the better test. All it does is compare checksums.

So having VERIFY on causes DOS to perform a partial test of the data on the disk after a write operation. This partial test will catch some, but not all, errors. One thing `VERIFY=ON` will always do is slow down disk operations. After writing data to disk, DOS must wait for that data area of the disk to rotate back under the head, read in the data, compute a new CRC, and compare it to the existing CRC. All that takes time. So having verify on catches some disk errors but slows down all disk write operations.

You must decide whether or not to have this tradeoff. My recommendation is to leave VERIFY on (`VERIFY=ON`). Entering VERIFY alone displays its current status.

MODE

The program MODE.COM is used to configure the serial ports on a PC. For example, in order to use my modem at 2400 baud, I have to issue the command:

```
MODE COM1: 2400,,,,,
```

Because I always use my modem at 2400 baud, I have this command in my AUTOEXEC.BAT file. Serial printers and plotters typically require a similar command or set of commands. You would also place these in the AUTOEXEC.BAT file. Note that MODE.COM must be in your path statement, and you must have already issued the PATH command or you need to specify the full path to MODE.COM, like this:

```
C:\DOS\MODE COM1: 2400,,,,,
```

Memory-resident software

Memory-resident software (also called TSR for terminate and stay resident) is software that stays in memory until you reboot or specifically remove it. There are special problems associated with loading memory-resident software. Generally, the software is only loaded once. Trying to load it a second time when it's already in memory can cause problems. If you load more than one memory-resident software program, there's usually only one loading order that will work.

The problems associated with memory-resident software make the AUTOEXEC.BAT file an excellent way to load it. With the AUTOEXEC.BAT file, you can specify a specific order so that you don't have to worry about loading in the wrong order. Because the AUTOEXEC.BAT file automatically loads the programs into memory when the computer is booted, they're always available without you having to remember to load them.

Communicating with the CONFIG.SYS file under DOS 6

As explained in the last chapter, DOS 6 allows the use of menus in the CONFIG.SYS file. When a menu is used, it automatically constructs an environmental variable called CONFIG that contains the name of the block selected by the user. In addition, the SET command can be used in the CONFIG.SYS file to create additional environmental variables to further document the user's selections. The AUTOEXEC.BAT file can test on these environmental variables and so only load certain software or take certain actions for particular configurations. This process was explained in the last chapter.

AUTOEXEC.BAT arrangements

As you have seen, the AUTOEXEC.BAT file has several important tasks:

- ❑ Perform some final configuration tasks not performed in the CONFIG.SYS file.
- ❑ Define a path.
- ❑ Create any other environmental variables required by the system. Some environmental variables, like COMSPEC, are created automatically. Others might be created in the CONFIG.SYS file.
- ❑ Define a prompt.
- ❑ Load any memory resident software.

There are two general approaches to performing these tasks, and both have advantages and disadvantages. One approach is to just perform everything in the AUTOEXEC.BAT file, while the other is to run other batch files to perform some or all of these tasks.

All-in-one approach

With the all-in-one approach to writing an AUTOEXEC.BAT file, you write one AUTOEXEC.BAT file to perform all the startup tasks you need performed. Using this approach, your AUTOEXEC.BAT file might look something like the one shown next:

```
@ECHO OFF
REM NAME:    AUTOEXEC.BAT
REM PURPOSE: Perform System Configuration All-In-One
REM VERSION: 4.01
REM DATE:    May 23, 1992
ECHO Setting Prompt and Environment
PROMPT=$e[s$e[1;1H$e[K$e[1B$e[K$e[1A$e[47m$e[34m$d$t$h$h
PROMPT=%PROMPT%$h$h$h$h  $e[1m$p$e[u$e[44m$e[37m$p$g
```

```
SET TEMP=D:\
SET GIFDESK=VGA
SET HARD=D:
SET FLOPPY=B:
PATH=C:\;..;;C:\NORTON;C:\SYSLIB;C:\UTILITY;C:\BAT;C:\WORD
PATH=%PATH%;C:\123;C:\VPIC;C:\ZIP;C:\LHA;D:\TODO;C:\BLDLITE
PATH=%PATH%;C:\NE;C:\DBASE
PATH=%PATH%
REM Delete .Tmp Files That Keep Appearing In Word
REM Subdirectory
IF EXIST C:\WORD\*.TMP DEL C:\WORD\*.TMP
REM Run Utility To Reduce Printer Retries From 20 To 1
NOWAIT-1    Uses LPT1
REM Run Utility To Turn Off Screen Print
NOPRTSCR
REM Loading CD ROM Driver
C:\CDROM\MSCDEX /D:MSCD210 /M:4 /L:E /V
C:\CDROM\MSCDINFO
IMAGE C: /SAVE
IMAGE D: /SAVE
LOCKEYS N:OFF
CD\
IF EXIST \WINDOWS\*.SWP DEL \WINDOWS\*.SWP > NUL
C:\WINDOWS\SMARTDRV
SCREEN1
```

This is not a made-up AUTOEXEC.BAT file. Rather, it is a slightly modified version of my own AUTOEXEC.BAT file from a few months ago. Only one modification was made, and I'll describe that in a moment.

The advantage of an all-in-one AUTOEXEC.BAT is that everything is collected into one location for easy maintenance. Plus, using a single file saves space and makes it easier to share a copy with someone else.

Remember: The all-in-one AUTOEXEC.BAT file saves disk space and keeps everything together. ☑

The first disadvantage of an all-in-one AUTOEXEC.BAT is easy to see. The AUTOEXEC.BAT file tends to become long, hard to follow, and hard to maintain. This problem can become much worse when you build menus into your CONFIG.SYS file to select between multiple configurations and then you modify your AUTOEXEC.BAT file to respond differently to these different configurations.

As you can see, when you boot with multiple configurations, multiple operating systems, or just have a lot of things to do in the AUTOEXEC.BAT file, it quickly becomes very confused. If you can spare the space for multiple batch files, the task of writing an AUTOEXEC.BAT file for all these conditions can be greatly simplified.

One-per-task approach

I mentioned earlier that my AUTOEXEC.BAT file shown earlier had one slight modification. My real AUTOEXEC.BAT file is much shorter and calls several other batch files to perform much of the work. In order to illustrate an all-in-one

AUTOEXEC.BAT file, I simply merged all these together. My actual working AUTOEXEC.BAT file is shown next:

```
@ECHO OFF
REM NAME:    AUTOEXEC.BAT
REM PURPOSE: Perform System Configuration All-In-One
REM VERSION: 4.01
REM DATE:    May 23, 1992
ECHO Setting Prompt and Environment
CALL C:\BAT\NICEPROM
CALL C:\BAT\SETS
CALL C:\BAT\SETPATH
REM Delete .Tmp Files That Keep Appearing In Word
REM Subdirectory
IF EXIST C:\WORD\*.TMP DEL C:\WORD\*.TMP
REM Run utility to reduce printer retries from 20 to 1
NOWAIT-1    Uses LPT1
REM Run Utility To Turn Off Screen Print
NOPRTSCR
REM Loading CD ROM Driver
CALL C:\BAT\CDROM
IMAGE C: /SAVE
IMAGE D: /SAVE
LOCKEYS N:OFF
CD\
FASTCONS
IF EXIST \WINDOWS\*.SWP DEL \WINDOWS\*.SWP > NUL
C:\WINDOWS\SMARTDRV
SCREEN1
```

While this version is still not short or simple, it is much simpler than the all-in-one version shown previously.

Remember: With a one-per-task AUTOEXEC.BAT file, the AUTOEXEC.BAT file calls other batch files to perform a number of tasks.

Besides making the AUTOEXEC.BAT shorter and easier to understand, this one-per-task approach has another advantage. Your AUTOEXEC.BAT performs two different types of tasks:

- Tasks you would never want to perform again until you reboot, like loading memory resident software.
- Tasks you might need to perform again, like setting the path or prompt.

When all these tasks are performed in a single AUTOEXEC.BAT file, it can be difficult to refresh the path if you've made changes to it because running your AUTOEXEC.BAT file again will load your memory resident software again—at least unless you have some clever programming in your AUTOEXEC.BAT.

However, if you use the one-per-task approach, it's easy to refresh your path or return the prompt to its default value. These activities are performed by stand-alone

batch files that the AUTOEXEC.BAT file calls as needed. Because they are stand-alone batch files, you can run them from the command line.

This is especially easy if you write these stand-alone batch files to facilitate this. My SETPATH.BAT batch file is shown next:

```
@ECHO OFF
REM NAME:    SETPATH.BAT
REM PURPOSE: Set The Path
REM VERSION: 2.00
REM DATE:    May 31, 1992
ECHO SETTING PATH
PATH;
PATH=C:\
PATH=%PATH%;..
PATH=%PATH%;C:\NORTON
PATH=%PATH%;C:\SYSLIB
PATH=%PATH%;C:\UTILITY
PATH=%PATH%;C:\BAT
PATH=%PATH%;C:\WORD
PATH=%PATH%;C:\123
PATH=%PATH%;C:\VPIC
PATH=%PATH%;C:\ZIP
PATH=%PATH%;C:\LHA
PATH=%PATH%;D:\TODO
PATH=%PATH%;C:\BLDLITE
PATH=%PATH%;C:\NE
PATH=%PATH%;C:\DBASE
```

It begins by resetting the path to NUL. That way, any existing path will not interfere with the new path it's creating. Thus, it resets the path to the default even if I'm running it from the command line and a path already exists.

Protecting your AUTOEXEC.BAT file from installation programs

A few nasty installation programs alter either your CONFIG.SYS file or your AUTOEXEC.BAT file or even both. Some of them even do so without first asking permission!

You can fight back—especially with your AUTOEXEC.BAT file.

One thing you should do with both files to protect yourself is keep a backup copy somewhere safe so you can easily restore them if they are altered.

You can also try making them read-only with the commands:

```
ATTRIB +R C:\AUTOEXEC.BAT
ATTRIB +R C:\CONFIG.SYS
```

However, some installation programs will abort when they try to write to one of these files and DOS will not let it.

For your AUTOEXEC.BAT file, you can design absolute protection in two easy steps:

1. Rename your AUTOEXEC.BAT file to START.BAT or any other batch file name you want.
2. Create a one-line AUTOEXEC.BAT file with the command `@START.BAT` (where START.BAT should be whatever you renamed your original AUTOEXEC.BAT file).

While an installation program might still alter your one-line AUTOEXEC.BAT file, you can examine those changes and see which ones you want to keep, plus a short and simple `COPY CON` command can restore your AUTOEXEC.BAT file to its original one-line self.

WINSTART.BAT

Normally, you think about loading your memory resident programs in your AUTOEXEC.BAT file. However, some of those programs are programs—like a fax modem driver—that you will only use while in Windows. Before you load Windows, those programs just waste memory. If you run Windows in 386-enhanced mode, there is a better way.

While loading, Windows will process the file WINSTART.BAT if it finds it in the Windows subdirectory. You'll have to create it if you want it because the Windows installation program does not. Any memory resident programs you install here will be available while Windows is running but not before.

LEAVING.BAT

DOS automatically runs AUTOEXEC.BAT when you turn the computer on and that gives you a place to perform those tasks you need to perform when the computer is first turned on. Wouldn't it be nice if there was a batch file called something like LEAVING.BAT that would run automatically when you turned the computer off? That would give you a place to perform those tasks that need to be performed prior to turning off the computer—tasks like

- ❑ saving the contents of any RAM disks back to the hard disk.
- ❑ making an incremental backup.
- ❑ parking the hard disk heads.

Of course, the specific tasks that need to be performed on your system will be different. For example, on my system I don't use a RAM disk, so I can skip the first one; and the heads on my hard drive automatically park when the power is turned off, so I can skip the third step. However, I do need to eject any disk in my CD ROM drive, so LEAVING.BAT for my system might look like this:

```
XCOPY D:\*.* A: /S /M
C:\CDROM\EJECT
```

You will need to add the commands your system uses to park the heads. The manual that comes with your computer will tell you how to do this. Also, if you save data files to a RAM disk, you will need to add commands to copy them back to the hard disk. If you put working files on the RAM disk, like the Windows swap file, you can skip this.

Because DOS will not automatically run LEAVING.BAT before shutting off power to the computer, you must remember to do this. However, if you get into the habit, copying RAM disk information to the hard disk and performing an incremental backup will protect your data; and parking the heads on your hard disk will save wear and tear on your system.

Remember: A batch file like LEAVING.BAT will protect your data and save wear and tear on your system.

Summary

If an AUTOEXEC.BAT file exists in the root directory of the drive you boot from, DOS runs it as the last step when booting.

If there is an AUTOEXEC.BAT file to run, DOS does not prompt you for the date and time.

The AUTOEXEC.BAT file is a good place to create a path.

The AUTOEXEC.BAT file is a good place to create a prompt.

The AUTOEXEC.BAT file is a good place to set `VERIFY=ON`, issue any required MODE commands, and perform any other configuration tasks that cannot be performed in the CONFIG.SYS file.

The AUTOEXEC.BAT file is a good place to load memory resident software.

The AUTOEXEC.BAT file can communicate with the CONFIG.SYS file using environmental variables the CONFIG.SYS file creates. This allows the AUTOEXEC.BAT file to respond differently to different selections the user makes in the CONFIG.SYS file.

The AUTOEXEC.BAT file can be written as one large all-in-one batch file or a one-per-task batch file that calls other batch files to perform specific tasks.

The all-in-one arrangement saves disk space but leads to a more complex AUTOEXEC.BAT file.

If Windows finds the batch file WINSTART.BAT in the Windows subdirectory, it runs it while Windows is running. That makes WINSTART.BAT the perfect place to load memory resident programs that are only used while you are in Windows.

A batch file like LEAVING.BAT allows you to perform all those tasks you really should perform before turning off the computer.

Appendices
Reference

The appendices in this section are designed to give you quick access to a large amount of information. Often, this information is presented in the very briefest of fashions. For more information, refer back to earlier sections of this book or to your DOS manual.

Solutions to problems This appendix presents a solution to each of the problems in the book. These are not the only solutions, and your solution might even be better than the one given! — *Appendix A*

Batch file documentation This appendix provides brief documentation on the batch files that are included on the disk that comes with this book. — *Appendix B*

DOS reference This appendix provides a brief summary of the DOS commands. This can be useful as a quick and handy reference tool while writing batch files. If you need more information, consult your DOS manual or use the HELP command from the command line to use the DOS 6 HELP program. — *Appendix C*

A Solutions to problems

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: FORMAT.COM`
`REM PURPOSE: Format Diskette In A-Drive`
`REM VERSION: 1.00`
`REM DATE: May 12, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`CLS`	Clear the screen.

`ECHO PURPOSE`
`ECHO`
`ECHO You Are About To Format A Disk.`
`ECHO This Prepares The Disk To Be Used`
`ECHO By The Computer But It Also Erases`
`ECHO Any Information Already On The Disk.`
`ECHO`
	Explain the purpose to the user. Note that the lines that appear to be an Echo command by itself also contain the invisible Alt-255 character.

`ECHO What To Do`
`ECHO`
`ECHO Place The Disk To Be Formatted`
`ECHO In The A-Drive (The One On Top) And`
`ECHO Press Any Key. If You Are Not Sure`
`ECHO You Want To Erase This Disk, Press`
`ECHO Ctrl-Break And Answer Yes To Stop`
`ECHO This Batch File.`
`ECHO`
	Tell the user what to do.

`ECHO Safety Check`
`ECHO`
`ECHO If You Break Out Of This Batch`
`ECHO File, You Can Check The Disk To See`
`ECHO If It Has Any Files With The DIR A:`
`ECHO Command From The DR DOS Prompt.`
`ECHO`
	Tell the user what to do if he encounters a problem.

A-1
Answer to Problem 1. FORMAT.BAT gives you some general information on formatting a disk and then pauses before the actual formatting in case the user wants to press Ctrl-Break.

A-1
Continued.

Batch File Line	Explanation
`PAUSE`	Pause the batch file prior to running the formatting program to give the user a chance to read the screen and abort the process if desired.
`XYZ A:`	Perform the formatting using the renamed XYZ.COM.
`GOTO END`	Jump to the end of the batch file.

`:HELP`
`  ECHO Formats A Floppy Disk`
`  Echo In The A-Drive`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

A-2
Answer to Problem 2. AUTO-1.BAT is a sample AUTOEXEC.BAT file that performs the four tasks outlined in the book. In order for this file to work properly, you would have to rename it AUTOEXEC.BAT and store it in the root directory of the boot drive.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME:  AUTO-1.BAT`
`REM  Of Course, To Run`
`REM  Properly, Would Need`
`REM  To Be Renamed To`
`REM  AUTOEXEC.BAT`
`REM PURPOSE: Solution To Problem #2`
`REM VERSION: 1.00`
`REM DATE:  May 12, 1992`
	Documentation remarks.

`REM Delete These Tests After`
`REM Renaming To AUTOEXEC.BAT`
`IF NOT (%0)==() ECHO Do Not Run From`
`  Command Line`
`IF NOT (%0)==() ECHO Sample AUTOEXEC`
`  .BAT File`
`IF NOT (%0)==() GOTO END`
	Warning remarks in case the user runs from command line. Will display a warning and drop out to DOS.

`REM Set Path` `REM ========`	Documentation remarks for this section.
`ECHO Creating Path`	Tell the user what will happen next.
`PATH=C:\;C:\DRDOS;C:\BAT;C:\NORTON;` `  C:\123;C:\WP;C:\UTILITY;C:\MENU`	Set the path using a DR DOS command. Of course, in the batch file this is all on one line.
`REM Create Environmental Variables` `REM ==========================`	Documentation remarks for this section.
`ECHO Creating Environmental Variables`	Tell the user what will happen next.

`SET TEMP=C:\TEMP`
`SET DOSONLY=YES`
`SET LIB=D:\BLDLITE`
`SET OBJ=D:\BLDLITE`
	Create four environmental variables and store values to them.

Batch File Line	Explanation
`REM Turn Off Screen Prints` `REM ======================`	Documentation remarks for this section.
`ECHO Turning Off Screen Prints`	Tell the user what will happen next.
`NOPRTSCR`	Run a utility program to turn off screen printing. Many such programs are available on networks such as CompuServe and PC-Link.
`REM Starting Menu` `REM ==============`	Documentation remarks for this section.
`ECHO Starting Menu`	Tell the user what will happen next.
`MENU`	Run the menu program.
`:END`	Label marking the end of the batch file.

A-2
Continued.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: ACCTHELP.BAT`
`REM PURPOSE: Provide Accounting Help`
`REM VERSION: 1.00`
`REM DATE: May 12, 1992`
	Documentation remarks.

`CLS`	Clear the screen.

`ECHO Introduction`
`ECHO`
`ECHO This Accounting System Uses Three`
`ECHO Different Programs To Manage Customer`
`ECHO Accounts. They Are:`
`ECHO 1. DAILY.EXE`
`ECHO 2. WEEKLY.EXE`
`ECHO 3. ANNUAL.EXE`
`ECHO`
	Display the first section of information. Note that the lines containing just an Echo command are followed by an Alt-255 character to display a blank line.

`ECHO Daily`
`ECHO`
`ECHO This Program Performs The Closing That`
`ECHO Is Required Each Day. It Can Be Run By`
`ECHO Any Clerk Using The General Clerical`
`ECHO Password. Note That It Must Be Run`
`ECHO After 4PM And Not Before.`
`ECHO`
	Display the next section of information.

`PAUSE`	Pause the program until the user presses a key. This gives the user time to read the screen.
`CLS`	Once the user presses a key, clear the screen.

A-3
Answer to Problem 3. ACCTHELP.BAT explains how to use three accounting programs stored on the computer.

A-3
Continued.

Batch File Line	Explanation
ECHO Weekly ECHO ECHO This Program Performs The Closing That ECHO Is Required Each Week. It Can Only Be ECHO Run By A Senior Clerk Or Above And ECHO Requires A Management-1 Password. It ECHO Must Be Run On The Last Working Day Of ECHO The Month And Must Be Run After ECHO Running The Daily Closing Program. ECHO	Display the next section of information.
ECHO Annual ECHO ECHO This Program Performs The Annual ECHO Closing Required At The End Of Each ECHO Year. It Can Only Be Run By A Branch ECHO Manager And Requires A Management-4 ECHO Password. It Must Be Run On The Last ECHO Day Of The Year After Running The Daily ECHO Closing Program. ECHO	Display the next section of information.
PAUSE	Pause the program until the user presses a key. This gives the user time to read the screen.
CLS	Once the user presses a key, clear the screen.
ECHO Backups ECHO ECHO Corporate Policy Requires Two Backups ECHO To Be Performed After A Weekly Closing. ECHO One Backup Must Be Sent To The IS ECHO Department At Headquarters (24/185) ECHO And The Other Retained Locally. ECHO Three Backups Are Required After An ECHO Annual Closing. In Addition To The IS ECHO And Local Copies, A Copy Is To Be ECHO Sent To The Legal Department (28/185). ECHO	Display the next section of information.
ECHO More Help ECHO ECHO If You Need More Help, Contact: ECHO ECHO Mark Williams ECHO 24/185 ECHO Extension 256 ECHO	Display the last section of information.
PAUSE	Pause the program until the user presses a key.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: RUNCHECK.BAT`
`REM PURPOSE: Run Check-It-Out Program`
`REM And Report Errors`
`REM Solution To Problem #4`
`REM VERSION: 1.00`
`REM DATE: May 17, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`CLS`	Clear the screen.
`:TOP`	Label marking the top of a loop.
`CHKITOUT`	Run the Check-It-Out program.
`IF ERRORLEVEL 1 ECHO Check-It-Out Had Problems` `IF ERRORLEVEL 1 GOTO TOP`	If it had a problem, display a message and continue looping.
`ECHO Check-It-Out Ran Fine` `GOTO TOP`	If it did not have a problem, it reaches this point so display a message and continue looping.

`:HELP`
`ECHO Continually Runs CHKITOUT`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

A-4
Answer to Problem 4. RUNCHECK.BAT continually runs a program called Check-It-Out and displays the results on the screen.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: BOOT-4-D.BAT`
`REM PURPOSE: Reboot Until D-Drive Works`
`REM Solution To Problem #5`
`REM VERSION: 1.00`
`REM DATE: May 17, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user requested help.
`IF EXIST D:\*.* GOTO END`	If any file on the D-drive exists, then the batch file does not need to do anything so jump to the end.

`IF NOT (%0)==() ECHO Do Not Run From Command Line`
`IF NOT (%0)==() ECHO Rename To AUTOEXEC .BAT`
`IF NOT (%0)==() GOTO END`
	Make sure the user does not run from command line—except to get help.

`BOOT`	Run a program to reboot the computer. Of course, this halts the execution of this batch file.

A-5
Answer to Problem 5. BOOT-4-D.BAT is called by the AUTOEXEC.BAT file and reboots the computer if the D drive is not working.

A-5
Continued.

Batch File Line	Explanation

:HELP
  ECHO Reboots If D-Drive
  ECHO Not Working
GOTO END
	Display a help screen and exit the batch file when the user requests help.

:END	Label marking the end of the batch file.

A-6
Answer to Problem 6. READFILE.BAT displays the files entered on the command line one at a time by typing them to the screen and using the MORE filter to display them one screen at a time.

Batch File Line	Explanation
@ECHO OFF	Turn command-echoing off.
REM NAME: READFILE.BAT REM PURPOSE: Display ASCII Files REM VERSION: 1.00 REM DATE: May 17, 1992	Documentation remarks.
NEEDHELP %0 %1 IF ERRORLEVEL 1 GOTO HELP	Use the NeedHelp utility program to check and see if the user requested help.
:TOP	Label marking the top of a loop.
CLS	Clear the screen.
IF (%1)==() GOTO END	If there are no more replaceable parameters, jump to the end of the batch file.
ECHO Displaying %1	Display the name of the next file to be displayed.
TYPE %1 | MORE	Type the file and use the More filter to display it one screen at a time.
SHIFT	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
PAUSE	Pause the program until the user presses a key.
GOTO TOP	Continue looping.
:HELP ECHO Display ASCII Files GOTO END	Display a help screen and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.

A-7
Answer to Problem 7. COPYTHEM.BAT copies all the files specified on the command line to the A drive only if that file does not already exist on the A drive.

Batch File Line	Explanation
@ECHO OFF	Turn command-echoing off.
REM NAME: COPYTHEM.BAT REM PURPOSE: Copy Files To A-Drive REM VERSION: 1.00 REM DATE: May 26, 1992	Documentation remarks.
NEEDHELP %0 %1 IF ERRORLEVEL 1 GOTO HELP	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.

Batch File Line	Explanation
`IF (%1)==() GOTO ERROR`	If the user did not request help and did not specify files to copy, jump to an error-handling section. This must come after running the NeedHelp routine since *COPYTHEM/?* will not have a %1 under MS-DOS.
`IF (%1)==(XYZ123ABC) GOTO SUB`	If the batch file is running as a subroutine, jump to the subroutine section.
`:TOP`	Label marking the top of the main loop.
`IF ERRORLEVEL 1 GOTO END`	If Xcopy reported an error via errorlevel in the subroutine, exit the batch file.
`IF (%1)==() GOTO END`	Once the batch file runs out of replaceable parameters, exit the batch file.
`FOR %%J in (%1) DO` `CALL COPYTHEM XYZ123ABC %%J`	Loop through all the files matching the current %1 replaceable parameter and call this batch file as a subroutine for each one. This is required to handle files individually when the user specifies a wildcard.
`SHIFT`	Move all the replaceable parameters down one level.
`GOTO TOP`	Continue looping.
`:SUB`	Label marking the top of the subroutine portion of the batch file.
`IF EXIST A:%2 ECHO %2 Already` `On A-Drive`	If the file is already on the A-drive, warn the user.
`IF EXIST A:%2 GOTO END`	If the file is already on the A-drive, exit the subroutine without copying the file.
`XCOPY %2 A:`	If the subroutine reaches this point, the file does not exist on the A-drive so copy it with Xcopy in order to use the errorlevel.
`IF ERRORLEVEL 1 ECHO Problem` `Copying %2`	If Xcopy reports an error via errorlevel, display a warning message.
`GOTO END`	This is the end of the subroutine so exit the batch file so control will return to the calling version of this batch file.

A-7
Continued.

A-7
Continued.

Batch File Line	Explanation

`:ERROR`
`  ECHO Must Speficy The Files To`
`  ECHO Copy On The Command Line`
`GOTO END`
	Display an error message and exit when the user does not enter any replaceable parameters.

`:HELP`
`  ECHO Copies The Files You Specify`
`  ECHO On The Command Line To The`
`  ECHO A-Drive Without Writing`
`  ECHO Over Any Files`
`GOTO END`
	Display help and exit when requested by the user.

`:END`	Label marking the end of the batch file.

A-8
Answer to Problem 8. When run with a command following its name, USEOVER.BAT stores that command to an environmental variable and then runs the command. When run alone, USEOVER.BAT runs the command stored in the environment.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME:  USEOVER.BAT`
`REM PURPOSE: Use A Command Over And Over`
`REM VERSION: 1.02`
`REM DATE:  August 3, 1991`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO JUSTCMD`	If the user did not enter a new command, rerun the one already in the environment.
`REM Construct Command in` `REM Environmental Variable`	Documentation remarks.
`SET COMMAND=`	If the batch file reaches this point, then the user has entered a new command so delete the old one. This command has no effect if an old value does not exist.
`:TOP`	Label marking the top of the loop to define the command.
`  SET COMMAND=%COMMAND% %1`	Replace the contents of the Command environmental variable with its current value plus %1 added to the end. Building the final variable in this fashion avoids having to limit the command to nine components.

Batch File Line	Explanation
`SHIFT`	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
`IF (%1)==() GOTO JUSTCMD`	If no more portions of the command exist, jump to a section to run the command.
`GOTO TOP`	If more portions of the command exist, loop through again and add them.
`:JUSTCMD`	Beginning of the section to run the command.
`%COMMAND%`	Run the command.
`GOTO END`	Exit the batch file.

`:HELP`
`ECHO JUSTCMD Runs Stored Command`
`ECHO JUSTCMD Part1 Part2 ... PartN`
`ECHO Stores And Runs The New Command`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END`	Label marking the end of the batch file.

A-8
Continued.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME: COPYTHE2.BAT`
`REM Modified For Problem 9`
`REM And Stored As`
`COPYTHE2.BAT`
`REM PURPOSE: Copy Files To A-Drive`
`REM VERSION: 2.00`
`REM DATE: May 26, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO ERROR`	If the user did not request help and did not specify files to copy, jump to an error-handling section. This must come after running the NeedHelp routine since *COPYTHEM/?* will not have a %1 under MS-DOS.
`IF (%1)==(XYZ123ABC) GOTO SUB`	If the batch file is running as a subroutine, jump to the subroutine section.
`:TOP`	Label marking the top of the main loop.

A-9
Answer to Problem 9. COPYTHE2.BAT copies all the files specified on the command line to the A drive. Unlike COPYTHEM.BAT, COPYTHE2.BAT will ask the user what to do if there is already a version of the file on the A drive.

Batch File Line	Explanation
`IF ERRORLEVEL 1 GOTO END`	If Xcopy reported an error via errorlevel in the subroutine, exit the batch file.
`IF (%1)==() GOTO END`	Once the batch file runs out of replaceable parameters, exit the batch file.
`FOR %%J in (%1) DO` `  CALL COPYTHE2 XYZ123ABC %%J`	Loop through all the files matching the current %1 replaceable parameter and call this batch file as a subroutine for each one. This is required to handle files individually when the user specifies a wildcard.
`SHIFT`	Move all the replaceable parameters down one level and make available another replaceable parameter as %9.
`GOTO TOP`	Continue looping.
`:SUB`	Label marking the top of the subroutine portion of the batch file.
`IF EXIST A:%2 GOTO DUP`	If the file is already on the A-drive, jump to the subroutine to handle this.
`XCOPY %2 A:`	If the subroutine reaches this point, the file does not exist on the A-drive so copy it with Xcopy in order to use the errorlevel.
`IF ERRORLEVEL 1 ECHO Problem` `  Copying %2`	If Xcopy reports an error via errorlevel, display a warning message.
`GOTO END`	This is the end of the subroutine so exit the batch file so control will return to the calling version of this batch file.
`:DUP`	Beginning of the subroutine to deal with the file already existing on the A-drive.
`ECHO %2 Already Exists On A-Drive`	Tell the user about the duplicate file.

`ECHO A-Drive`
`ECHO --------`
`DIR A:%2`
	Show the user the file on the A-drive.

`ECHO Current Drive`
`ECHO -------------`
`DIR %2`
	Show the user the file on the current drive.

A-9
Continued.

Batch File Line	Explanation
`BATCMD YN Overwrite %2 On A-Drive (Y/N)?`	Ask the user if the batch file should overwrite the file on the A-drive.
`IF NOT ERRORLEVEL 1 GOTO END`	If the user answers no, exit the subroutine.
`XCOPY %2 A:`	The subroutine only reaches this point if the user answers yes so copy the file.
`IF ERRORLEVEL 1 ECHO Problem Copying %2`	If Xcopy reports an error via errorlevel, display a warning message.
`GOTO END`	This is the end of the subroutine so exit the batch file so control will return to the calling version of this batch file.

`:ERROR`
`  ECHO Must Speficy The Files To`
`  ECHO Copy On The Command Line`
`GOTO END`
	Display an error message and exit when the user does not enter any replaceable parameters.

`:HELP`
`  ECHO Copies The Files You Specify`
`  ECHO On The Command Line To The`
`  ECHO A-Drive Without Writing`
`  ECHO Over Any Files`
`GOTO END`
	Display help and exit when requested by the user.

`:END`	Label marking the end of the batch file.

A-9
Continued.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME:  PHONE.BAT`
`REM PURPOSE: Telephone Database`
`REM VERSION: 1.00`
`REM DATE:  June 11, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO NONE`	If the user did not enter a nickname, jump to an error-handling section.
`GOTO %1`	Jump to the section for the nickname the user entered. If the nickname entered by the user does not exist as a label, the batch file will abort on this line.

A-10A
Answer to Problem 10. PHONE.BAT dials the phone number for the nickname entered on the command line. You will have to enter your own nicknames and phone numbers before using PHONE.BAT.

A-10A
Continued.

Batch File Line	Explanation

`REM ----------------------------`
`REM The GOTO %1 Line Above Will`
`REM Cause The Batch File To`
`REM Abort If The Label Does`
`REM Exist`
`REM ----------------------------`
	Documentation remarks.

`:HELP`
`  ECHO Dials Phone`
`  ECHO The Syntax Is`
`  ECHO PHONE Nickname`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:NONE`
`  ECHO You Did Not Enter A`
`  ECHO Nickname To Dial`
`  ECHO The Syntax Is`
`  ECHO PHONE Nickname`
`GOTO END`
	Error-section that handles the situation with the user does not enter a nickname.

`:DIRECTORY`
`:INFORMATION`
`  ECHO ATDT 411 > COM1`
`  ECHO Dial Information`
`  ECHO Press Any Key When Ringing`
`  PAUSE`
`  ECHO ATH > COM1`
`GOTO END`
	Section that handles dialing information. Notice the use of multiple labels. The echoing of ATDT and ATH is explained in the text.

`:TAB`
`:TABBOOKS`
`  ECHO ATDT 1 (800) 233-1128 > COM1`
`  ECHO Dialing Tab Books`
`  ECHO Press Any Key When Ringing`
`  PAUSE`
`  ECHO ATH > COM1`
`GOTO END`
	Section that handles dialing Tab Books.

`:911`
`  ECHO ATDT 911 > COM1`
`  ECHO 911`
`  ECHO Press Any Key When Ringing`
`  PAUSE`
`  ECHO ATH > COM1`
`GOTO END`
	Section that handles dialing 911.

`:END`	Label marking the end of the batch file.

A-10B
Alternate answer to Problem 10. PHONE1.BAT does everything PHONE.BAT does, plus built-in error-checking. Before issuing a GOTO %1 command, it searches PHONE1.BAT to ensure that the specified batch file exists. While slower and more complex, this check prevents the batch file from aborting with the "Label not found" error message.

Batch File Line	Explanation
`@ECHO OFF`	Turn command-echoing off.

`REM NAME:  PHONE1.BAT`
`REM PURPOSE:  Telephone Database`
`REM  Modified Version Of`
`REM  PHONE.BAT That Checks`
`REM  For Valid Label First`
`REM VERSION:  1.00`
`REM DATE:  June 11, 1992`
	Documentation remarks.

`NEEDHELP %0 %1` `IF ERRORLEVEL 1 GOTO HELP`	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
`IF (%1)==() GOTO NONE`	If the user did not enter a nickname, jump to an error-handling section.
`REM Type The Batch File And Search` `REM For A Valid Label`	Documentation remarks.

Batch File Line	Explanation
`TYPE C:\BAT\PHONE1.BAT|FIND ":%1">JUNK`	Type the batch file and use the Find filter to search for the specified label. Notice that the full path to the batch file is specified. This is required since you never know from which subdirectory it will be run. Also note the colon before the %1, this makes sure the matching text is a label.
`ISITZERO JUNK`	Check to see if the resulting file is a 0-length file.
`IF NOT ERRORLEVEL 4 GOTO %1`	IsItZero sets errorlevel to four for a zero length file and that indicates the label was not found. For any other errorlevel value, jump to the appropriate section.
`GOTO MISSING`	Jump to an error-handling section that handles the problem of the user entering an invalid label.

`:MISSING`
`  ECHO The Label You Specified (%1)`
`  ECHO Does Not Exist As A Nickname`
`  ECHO PHONE1.BAT Aborting`
`GOTO END`
	When the label is not found, explain the problem to the user and exit the batch file.

`:HELP`
`  ECHO Dials Phone`
`  ECHO The Syntax Is`
`  ECHO PHONE Nickname`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help. From here on down, PHONE1.BAT is identical to PHONE.BAT.

`:NONE`
`  ECHO You Did Not Enter A`
`  ECHO Nickname To Dial`
`  ECHO The Syntax Is`
`  ECHO PHONE Nickname`
`GOTO END`
	Error-section that handles the situation with the user does not enter a nickname.

`:DIRECTORY`
`:INFORMATION`
`  ECHO ATDT 411 > COM1`
`  ECHO Dial Information`
`  ECHO Press Any Key When Ringing`
`  PAUSE`
`  ECHO ATH > COM1`
`GOTO END`
	Section that handles dialing information. Notice the use of multiple labels. The echoing of ATDT and ATH is explained in the text.

`:TAB`
`:TABBOOKS`
`  ECHO ATDT 1 (800) 233-1128 > COM1`
`  ECHO Dialing Tab Books`
`  ECHO Press Any Key When Ringing`
`  PAUSE`
`  ECHO ATH > COM1`
`GOTO END`
	Section that handles dialing Tab Books.

A-10B
Continued.

A-10B
Continued.

Batch File Line	Explanation
:911 ECHO ATDT 911 > COM1 ECHO 911 ECHO Press Any Key When Ringing PAUSE ECHO ATH > COM1 GOTO END	Section that handles dialing 911.
:END	Label marking the end of the batch file.

A-11
Answer to Problem 11. SHOWBACK.BAT lists every file on the C drive that has been modified since the last backup.

Batch File Line	Explanation
@ECHO OFF	Turn command-echoing off.
REM NAME: SHOWBACK.BAT REM PURPOSE: Show Files Needing Backup REM VERSION: 1.00 REM DATE: June 11, 1992	Documentation remarks.
NEEDHELP %0 %1 IF ERRORLEVEL 1 GOTO HELP	Use the NeedHelp utility program to check and see if the user requested help.
ECHO ---WORKING--- ECHO ---This Takes Time---	Tell the user what is happening.
ATTRIB C:*.* /S | FIND " A " | MORE	Find the files needing backup and display them with the MORE filter.
GOTO END	Exit the batch file.
:HELP ECHO Shows Files Needing Backup GOTO END	Display a help screen and exit the batch file when the user requests help.
:END	Label marking the end of the batch file.

A-12
Answer to Problem 12. QCD.BAT allows the user to enter a drive and subdirectories after the batch file name without a colon or backslash, and it still changes to the proper subdirectory.

Batch File Line	Explanation
@ECHO OFF	Turn command-echoing off.
REM NAME: QCD.BAT REM PURPOSE: Speed Directory Changes REM VERSION: 1.01 REM DATE: December 20, 1991 REM REVISION: 1.01 Added Deleting NeedSpace So Program Would Work When Shelled Out	Documentation remarks.
NEEDHELP %0 %1 IF ERRORLEVEL 1 GOTO HELP	Use the NeedHelp utility program to check and see if the user requested help.
IF (%1)==() GOTO ERROR1	The user did not specify a subdirectory so jump to an error section.

IF (%1)==(\) CD\
IF (%1)==(\) SHIFT
IF (%1)==() GOTO END
	If the user requested the root directory, change to there and exit.

Batch File Line	Explanation
`SET NEEDSPACE=`	Reset this environmental variable to make space in case QCD.BAT is run while shelled out of another program. On my system, "Needspace" contains a long string of X's.
`SET DIR=NO`	Create a flag environmental variable.
`FOR %%J IN (a A b B c C d D) DO` `  IF (%1)==(%%J) SET DIR=YES`	Loop through all the drives to see if the user is changing drives. If so, change a flag.
`IF %DIR%==NO GOTO SKIP`	If the flag indicates no drive change, skip drive change code.
`%1:`	Change to the requested drive.
`SHIFT`	Remove the drive replaceable parameter.

`IF (%1)==(\) CD\`
`IF (%1)==(\) SHIFT`
`IF (%1)==() GOTO END`
	Check to see if the next replaceable parameter requests a change to the root directory and if so change there.

`:SKIP`	Label to skip drive change.
`SET DIR=`	Remove the drive change flag environmental variable.
`:TOPLOOP`	Top of loop to create environmental variable containing subdirectory.
`  SET DIR=%DIR%\%1`	Add a subdirectory.
`  SHIFT`	Move all the replaceable parameters down one level.
`  IF NOT (%1)==() GOTO TOPLOOP`	If there are more replaceable parameters, continue looping.
`  CD %DIR%`	Change to the requested subdirectory.
`GOTO END`	Exit the batch file.

`:ERROR`
`  ECHO No Drive / Subdirectory`
`  Specified`
`  ECHO Use QCD Drive Sub1 Sub2 So On`
`  ECHO Without A Colon Or Backslash`
`GOTO END`
	Section to display an error message.

`:HELP`
`  ECHO QCD.BAT Will Change Drives And`
`  ECHO Subdirectories Without`
`  Entering`
`  ECHO The Colon After The Drive Or`
`  The`
`  ECHO Backslash Between The`
`  ECHO Subdirectories`
`  BATCMD SL`
`  ECHO Enter: QCD Drive Sub1 Sub2 So`
`  On`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

A-12
Continued.

A-12
Continued.

Batch File Line	Explanation
:END	Label marking the end of the batch file.
SET DIR=	Reset the environmental variable used by the batch file.
SET NEEDSPACE=XXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXX	Recreate the environmental variable that is used to free up environmental space when the batch file is run while shelled out.

A-13
Answer to Problem 13. KILLZERO.BAT will list the 0- length files on the screen and give you the option to delete them. If you select "Yes," it automatically deletes them all.

Batch File Line	Explanation
@ECHO OFF	Turn command-echoing off.
REM NAME: KILLZERO.BAT REM PURPOSE: Erase 0-Length Files REM VERSION: 1.00 REM DATE: June 11, 1992	Documentation remarks.
IF (%1)==(ABC123XYZ) GOTO SUB1 IF (%1)==(XYZ123ABC) GOTO SUB2	If KILLZERO.BAT is being called as a subroutine, jump to the appropriate subroutine section.
NEEDHELP %0 %1 IF ERRORLEVEL 1 GOTO HELP	Use the NeedHelp utility program to check and see if the user started the batch file with a /? switch.
SET KILL=NO	Create an environmental variable to act as a flag indicating if any 0-length files are found.
FOR %%J IN (*.*) DO CALL KILLZERO ABC123XYZ %%J	Loop through the files once and call KILLZERO.BAT as a subroutine to list the 0-length files.
IF (%KILL%)==(NO) ECHO No 0-Length Files To Erase IF (%KILL%)==(NO) GOTO END1	If no 0-length files were found, display a message and jump to an exiting section.
BATCMD YN Erase These Files? (Y/N)	If the batch file reaches this point, 0-length files were found. Ask the user about deleting them.
IF NOT ERRORLEVEL 1 GOTO END1	If the user answers no, jump to an exiting section.
FOR %%J IN (*.*) DO CALL KILLZERO XYZ123ABC %%J	If the batch file reaches this point, the user elected to delete the 0-length files. Call KILLZERO.BAT as a subroutine to do that.
GOTO END1	Jump to an exiting section.
:SUB1	Label marking the beginning of the first subroutine.

Batch File Line	Explanation
`ISITZERO %2`	The file name is passed as the second replaceable parameter. Test it to see if it is a 0-length file.
`IF ERRORLEVEL 4 ECHO %2 To Be Erased`	If it is, display its name.
`IF ERRORLEVEL 4 SET KILL=YES`	If it is a 0-length file, set a flag.
`GOTO END`	Exit the subroutine.

`:SUB2`
`ISITZERO %2`
`IF ERRORLEVEL 4 ECHO Killing %2`
`IF ERRORLEVEL 4 ERASE %2`
`GOTO END`
	Use the same procedures as the first subroutine to identify and delete the 0-length files. Users should be careful with this section. A mistake can cause all your files to be erased.

`:HELP`
`ECHO Automatically Erase 0-Length Files`
`GOTO END`
	Display a help screen and exit the batch file when the user requests help.

`:END1`
`SET KILL=`
`GOTO END`
	This exiting section is used to reset the environmental variable. This cannot be handled after the :END label since the subroutines use that label to exit and the value is still needed after the subroutines run.

`:END`	Label marking the end of the batch file.

A-13
Continued.

Batch file documentation

The following appendix is a quick reference to the sample batch files included on your disk. For more detailed information on these scripts, refer to the appropriate chapter.

Many of the batch files included in this book and on the disk are teaching batch files. In other words, the batch files were designed to teach concepts and not perform useful tasks. As a result, many of them will require modifications in order to be useful to you on a daily basis. In many cases, these modifications are quite minor.

You might find these batch files useful right away, without any modification:

A
ANSIKEY1
ANSIKEY2
BLANK
CATALOG
CHECK
CL
COPYTHEM
COPYTHE2
COUNTENV
EDIT2
EDITBAT
EDITPATH
ENTERHEX
FASTFIND

HELPBAT
HELPBAT3
I
KEYASSGN
KEYDEL
KEYERASE
KILLZERO
LISTFILE
MACRO
MOVEIT
MULTI
NICEPROM
NOBURN
QCD
READFILE
REMOVE
RETURN
RETURNTO
RETURN2
RETURN3
RONNYMD
SAYERROR
SENDANSI
SENDANS2
SHOWBACK
TEMPLATE
TOA
UNBLANK
UNENTER
USEOVER
WINDIR

Contents of \Bat subdirectory

The following batch files are located in the \BAT subdirectory on the disk that comes with this book.

0–9.BAT

Syntax 0-9

A demonstration batch file that changes to the C:\D\LOTUSFIG\0-9 subdirectory.

1.BAT through 6.BAT

Syntax 1 [/?]

These batch files are used to run the menu options used by the demonstration non-resident and hybrid menu systems on the disk in the subdirectories \MENU1 and \MENU2. They are designed to simulate applications without really running them.

A.BAT

Syntax A [/?]

A.BAT performs a directory of the A drive.

ACCOUNT1.BAT

Syntax `ACCOUNT1 [/?]`

A demonstration batch file that runs an accounting closing program and then tests the errorlevel in order to store information in the environment.

ACCOUNT2.BAT

Syntax `ACCOUNT2 [/?]`

A demonstration batch file that runs the accounting program CLOSE2 only after the batch file ACCOUNT1.BAT has run the accounting program CLOSE successfully. Demonstrates testing on the environment before running a program.

ACCTHELP.BAT

Syntax `ACCTHELP [/?]`

ACCTHELP.BAT is a demonstration batch file that displays imaginary help for three accounting problems. This is a solution to Problem 3 in the book.

ANSIDEMO.BAT

Syntax `ANSIDEMO [/?]`

ANSIDEMO.BAT is a demonstration batch file. While you might find it useful to include similar code in other batch files, ANSIDEMO.BAT performs no useful function beyond demonstrating how to include ANSI escape sequences in a batch file. Readers wanting to run ANSIDEMO.BAT must load ANSI.SYS in their CONFIG.SYS file.

ANSIHIDE.BAT

Syntax `ANSIHIDE [/?]`

ANSIHIDE.BAT is a demonstration that shows how ANSI escape sequences can change the foreground and background colors to black, effectively hiding messages that are displayed through DOS. While you might find it useful to include similar code in other batch files, ANSIHIDE.BAT performs no useful function beyond demonstrating how to hide DOS messages in a batch file.

ANSIKEY1.BAT

Syntax `ANSIKEY1 91 40 93 41 40 91 41 93 [/?]`

ANSIKEY1.BAT is used to reassign pairs of keys using their ASCII values. The first number is the key to reassign, and the second number is the ASCII value to assign to that key. As many pairs of numbers can be specified after the batch file name as will fit on the command line. The keyboard reassignment is performed using Ansi, so reassignments will not work in programs that bypass DOS to read the keyboard. ANSIKEY1.BAT is designed for experimentation. Users wanting to make the same reassignments on a regular basis should modify this batch file to issue those commands automatically without having to enter them on the command line.

ANSIKEY2.BAT

Syntax `ANSIKEY2 ( [ ) ] [ ( ] ) [/?]`

ANSIKEY2.BAT is used to reassign pairs of keys using their actual values. The first value is the key to reassign, and the second value is the value to assign to that key. The sample command shown here reassigns the parentheses key to square brackets

and the square brackets to parentheses. As many values can be specified after the batch file name as will fit on the command line. The keyboard reassignment is performed using ANSI, so reassignments will not work in programs that bypass DOS to read the keyboard. ANSIKEY2.BAT is designed for experimentation. Users wanting to make the same reassignments on a regular basis should modify this batch file to issue those commands automatically without having to enter them on the command line.

AUTO-1.BAT

Do not run AUTO-1.BAT from the command line. AUTO-1.BAT is a sample AUTOEXEC.BAT file that is the solution to Problem 2 in the book. Of course, before being used, it must be renamed to AUTOEXEC.BAT.

BLANK.BAT

Syntax `BLANK [/?]`

BLANK.BAT changes the foreground and background colors to black, effectively blanking the screen. Only programs that display their text through DOS will be blanked. Users wanting to run BLANK.BAT must load ANSI.SYS in their CONFIG.SYS file. UNBLANK.BAT can be run to restore the screen colors.

BOOT-4-D.BAT

Do not run BOOT-4-D from the command line. BOOT-4-D.BAT continually reboots the computer until the D drive is available. This is a solution to Problem 5 in the book. It must be called by the AUTOEXEC.BAT file in order to run properly.

CATALOG.BAT

Syntax `CATALOG [Diskette Name] [/?]`

CATALOG.BAT creates a catalog of the files on multiple floppy disks by echoing the name of the diskette entered on the command line and a directory of the floppy disk in the A drive to a file called C:\CATALOG\CATALOG.TXT. A name must be entered on the command line. If the name is longer than nine words, everything after the ninth word is ignored. REMOVE.BAT is used to remove entries.

CHECK.BAT

Syntax `CHECK [/?] [/F]`

Runs CHKDSK and passes it any CHKDSK switches entered on the command line. Will not run CHKDSK while shelled out from Windows to prevent the user running it with a /F switch.

CHECKERR.BAT

Syntax `CHECKERR [/?]`

CHECKERR.BAT is a demonstration batch file that displays the current errorlevel. CHECKERR.BAT is a large batch file that takes the brute-force approach of simply testing for all 256 possible errorlevel values.

CHECKER1.BAT

Syntax `CHECKER1 [/?]`

CHECKER1.BAT is a demonstration batch file that displays the current errorlevel. It uses nested If-tests to produce a batch file much shorter than CHECKERR.BAT.

CHECKWIN.BAT

Syntax CHECKWIN [/?]

This is a demonstration batch file that checks to see if it is running while shelled out of Windows without needing a utility program—although it does need the DOS FIND program. Requires Windows 3.1 to run properly. Because you don't know what utilities other users will have, this is a good way to check for Windows in batch files that either require Windows or will not run properly under Windows.

CL.BAT

Syntax CL *Command*
CL D
CL *n*
CL [/?]

CL.BAT is used to store long command lines, display the saved contents of command lines, and execute saved command lines. CL.BAT saves commands in the C:\CL subdirectories using the batch files 0.BAT through 9.BAT. It uses the first available batch file. If all of them are in use, it deletes 0.BAT, moves the other batch files down one level, and saves the command line in 9.BAT. After saving the command line, CL.BAT executes that command line.

The CL D command is used to display the saved command lines. The CL *n* command is used to run an existing command line where *n* is the number of the saved command line to run.

CL.BAT is useful even if you have DOSKEY. Because CL.BAT only saves those command lines you run using CL.BAT, you can use it only on your long command lines. As a result, you do not have to wade through all the shorter command lines saved by DOSKEY to find a long one to reuse. Additionally, CL.BAT is able to store command lines between sessions, something DOSKEY and other similar programs cannot do.

COMMANDS.BAT

Syntax COMMANDS *Commands* | [/?]

This is a demonstration batch file that shows using replaceable parameters as commands in a batch file.

COMPILE.BAT

Syntax COMPILE [/?]

This is a demonstration batch file that compiles a Quicksilver program using the name stored in the "COMPILE" environmental variable to supply the name of the program.

COPYBACK.BAT

Syntax COPYBACK *Source Target* [/?]

COPYBACK.BAT is used as part of the document archival system in Chapter 30. Copies an important file to a backup file name in the C:\OLD subdirectory. It checks first to make sure the Target filename does not already exist in the C:\OLD subdirectory.

COPYTHEM.BAT

Syntax COPYTHEM *File1 File2 ... FileLast* [/?]

COPYTHEM.BAT copies the files you specify on the command line to the A drive without writing over any files. This is the solution to Problem 7 in the book.

COPYTHE2.BAT

Syntax COPYTHE2 *File1 File2 ... FileLast* [/?]

COPYTHE2.BAT copies the files you specify on the command line to the A drive without writing over any files. Unlike COPYTHEM.BAT, COPYTHE2.BAT gives you the option of overwriting files when they exist on the A drive. This is the solution to Problem 9 in the book.

COUNTENV.BAT

Syntax COUNTENV [/?]

Displays the approximate amount of space being used by all your environmental variables.

DELOLD.BAT

Syntax DELOLD [/?]

DELOLD.BAT is used as part of the document archival system in Chapter 30. After copying files to the A drive from the \DISCARD subdirectory, DELOLD.BAT deletes the files in the C:\DISCARD that exists on the A drive.

DISCARD.BAT

Syntax DISCARD *File1 File2 ... FileLast* [/?]

DISCARD.BAT is used as part of the document archival system in Chapter 30. It copies all the files listed on the command line to the C:\DISCARD subdirectory. Prior to copying, it checks to make sure that a file doesn't already exist in the C:\DISCARD subdirectory with the same name. After successfully copying the file, it erases the original.

DOBATCMD.BAT

Syntax DOBATCMD [/?]

DOBATCMD.BAT is a demonstration batch file that illustrates most of the commands in Batch Commands (Batcmd) not involved in getting information from the user. SHOWGETS.BAT is a second demonstration batch file that illustrates using Batcmd to get information from the user.

DOXCOPY.BAT

Syntax DOXCOPY *Source Target Switches* [/?]

DOXCOPY.BAT runs XCOPY and reports any error conditions it encounters by testing on the errorlevel. The information entered after the DOXCOPY command should be identical to the information you would supply the Xcopy command.

EDIT2.BAT

Do Not Run From Command Line

EDIT2.BAT loops through each subdirectory in your path and asks you if you want to keep that subdirectory in your path. After processing all the subdirectories, it constructs a new path containing just the subdirectories you elected to keep. EDIT2.BAT is designed to be called by EDITPATH.BAT and should not be run from the command line. Doing so will reset your path to nul.

EDITBAT.BAT

Syntax `EDITBAT` *`Batch File`* `| [/?]`

This batch file lets you keep all your batch files in the C:\BAT subdirectory and still edit and create them from any subdirectory. It also automatically creates a backup file if you edit an existing batch file.

EDITPATH.BAT

Syntax `EDITPATH [/?]`

EDITPATH.BAT calls EDIT2.BAT and passes it the path as a series of replaceable parameters. EDIT2.BAT loops through each subdirectory in your path and asks you if you want to keep that subdirectory in your path. After processing all the subdirectories, it constructs a new path containing just the subdirectories you elected to keep. Run EDITPATH.BAT and not EDIT2.BAT, as EDIT2.BAT will not run properly from the command line.

ENDLESS.BAT

Syntax `ENDLESS [/?]`

ENDLESS.BAT is a demonstration batch file that illustrates the GOTO command with a loop that continues until you press Ctrl-Break and stop it.

ENTERHEX.BAT

Syntax `ENTERHEX [/?]`

Uses ANSI.SYS to convert the keys around the number pad to A–F for easier hexadecimal entry when using Edlin or "COPY CON." Does not work with Edit. The "/" is converted to an "A," the "*" is converted to a "B," the "–" is converted to a "C," the "+" is converted to a "D," the "=" is converted to an "E" and the Del key is converted to an "F."

FASTFIND.BAT

Syntax `FASTFIND` *`FileSpec`* `| [/?]`

FASTFIND.BAT searches the current hard disk for the file name specified on the command line. Wildcards are allowed.

FINDPASS.BAT

Syntax `FINDPASS [/?]`

Chapter 32 shows how to add password protection to a batch file using PASSWORD.BAT as an example. PASSWORD.BAT has you enter a three-digit password by entering the ASCII values of the three digits. Rather than having you

use an ASCII chart, FINDPASS.BAT prompts you for the password you want to use and displays the appropriate ASCII values on the screen.

FORMAT.BAT

Syntax `FORMAT [/?]`

FORMAT.BAT automatically formats the A drive even if the user fails to specify a drive. It also explains the command to the user. This is a solution to Problem 1 in the book.

HELP.BAT

Syntax `HELP [/?]`

HELP.BAT is a demonstration batch file that displays summary information about the function of several other batch files.

HELP1.BAT

Syntax `HELP1 Category | [/?]`

HELP1.BAT is a modified version of HELP.BAT that uses the category entered on the command line to jump to a specific category of batch file. This allows it to provide limited context-sensitive help.

HELPBAT.BAT

Syntax `HELPBAT [/?]`

HELPBAT.BAT uses the documentation information built into batch files—as described in this book—to display custom help about each batch file. It loops through each batch file in the C:\BAT subdirectory. For each batch file, it first displays the name of the batch file and underlines it. After displaying the name, it displays only the line containing "PURPOSE:." This allows the user to scroll through all her batch files easily, and review their function.

HELPBAT2.BAT

Syntax `HELPBAT2 [Topic Of Interest] | /?]`

HELPBAT2.BAT is a modified version of HELPBAT.BAT. It displays the name of all the batch files but only displays the purpose line for those batch files containing the words entered on the command line in the purpose line. The words must be in the same order and match exactly.

HELPBAT3.BAT

Syntax `HELPBAT3 [Topic Of Interest] | /?]`

HELPBAT3.BAT is a modified version of HELPBAT2.BAT that still searches for text but only displays the names of the batch files matching the purpose—along with the purpose line displayed just like HELPBAT2.BAT. This makes it much easier to find a batch file that performs a specific function because you don't have to see all the batch file names as part of the search—just the ones that match your search criteria. The words must be in the same order and match exactly.

HI-ASCII.BAT

Syntax `HI-ASCII [/?]`

HI-ASCII.BAT is a demonstration batch file that displays two types of boxes that can be created using high-ordered ASCII characters as well as a single type of box that can be created without using high-ordered ASCII characters.

I.BAT

Syntax `I [/?]`

This batch file performs an incremental backup of the C drive.

INDEX.BAT

Syntax `INDEX Number [/?]`

INDEX.BAT is a demonstration batch file that uses the ZyIndex program to index all the text files on the A drive. A number for the diskette must be specified after the INDEX command.

KEYASSGN.BAT

Syntax `KEYASSGN [/?]`

KEYASSGN.BAT performs a number of DOSKEY command reassignments to make DOS friendlier.

KEYDEL.BAT

Syntax `KEYDEL [Files | /?]`

KEYDEL.BAT was designed to be defined to the DOS DEL command by a DOSKEY macro. It lists all the files to be deleted and then asks permission. It can also be run as a stand-alone batch file with the files to be deleted supplied as the first replaceable parameter.

KEYERASE.BAT

Syntax `KEYERASE [Files | /?]`

KEYERASE.BAT was designed to be defined to the DOS ERASE command by a DOSKEY macro. It lists all the files to be deleted and then asks permission. It can also be run as a stand-alone batch file with the files to be deleted supplied as the first replaceable parameter.

KILLZERO.BAT

Syntax `KILLZERO [/?]`

KILLZERO.BAT erases all the 0-length files in the current subdirectory. This is the solution to Problem 13 in the book.

LISTFILE.BAT

Syntax `LISTFILE [/?]`

LISTFILE.BAT creates a list of all the files on the C drive and stores it in C:\CHANGED.TXT file. This is useful as a record of the current file names just prior to making an incremental backup.

MACRO.BAT

Syntax MACRO *n Command*
MACRO D
MACRO *n*
MACRO [/?]

MACRO.BAT is used to store keyboard macros, display the contents of the keyboard macros and execute the saved keyboard macros. MACRO.BAT saves commands in the C:\MACRO subdirectories using the batch files 0.BAT through 9.BAT. It saves the macro under the name n specified on the command line. After saving the macro, MACRO.BAT does not execute that macro. The MACRO D command is used to display the saved macros. The MACRO n command is used to run a macro where n is the number of the saved macro to run. Additionally, MACRO.BAT is able to store macros between sessions, something other keyboard macro programs cannot do.

MENU1.BAT

Syntax MENU1 [/?]

MENU1.BAT drives the non-resident non-nested demonstration menu system stored in the \MENU1 subdirectory on the disk that comes with this book.

MENU2.BAT

Syntax MENU2 [/?]

MENU2.BAT drives the hybrid non-nested demonstration menu system stored in the \MENU2 subdirectory on the disk that comes with this book.

MENU3.BAT

Syntax MENU3 [/?]

MENU3.BAT drives the resident non-nested demonstration menu system stored in the \MENU3 subdirectory on the disk that comes with this book. MENU3.BAT simulates applications rather than running actual applications.

MOVEIT.BAT

Syntax MOVEIT [*Source Target* | /?]

MOVEIT.BAT is designed to function as a DOSKEY macro to move files from one location (Source) to another (Target) safely. While designed as a DOSKEY macro, it will also run properly from the command line. With the addition of the MOVE command in DOS 6, this batch file is no longer needed.

MULTI.BAT

Syntax MULTI *Command1 ^ Command2 ^ ... ^ Last Command* [/?]

MULTI.BAT allows you to enter multiple commands on one command line and have them executed in turn. The commands must be separated by a caret (^) with a space on both sides of it. You can enter as many commands as will fit on a command line, and each command may have as many terms as necessary; however, piping is not allowed. MULTI.BAT is not needed while DOSKEY is running.

NICEPROM.BAT

Syntax `NICEPROM [/?]`

NICEPROM.BAT creates a very nice prompt. It displays the date, time, and current subdirectory at the top of the screen along with the drive and subdirectory in the usual prompt position. It also sets the screen colors to bright-white on blue. Users wanting to run NICEPROM.BAT must load ANSI.SYS in their CONFIG.SYS file.

NOBURN.BAT

Syntax `NOBURN [/?]`

This batch file clears the screen until you press any key, giving you a free screen saver. Its low memory requirements mean that you can run it while shelled out of another program.

OCCASION.BAT

Do Not Run From Command Line

OCCASION.BAT is designed to be called by the AUTOEXEC.BAT file and should not be run from the command line. Every tenth time the AUTOEXEC.BAT file calls it, OCCASION.BAT performs a backup. It tracks the number of times it runs by a series of If-tests that simply try all possible values for the name of the 0-length file used as a counter. Note that the batch file counts reboots and not days.

OCCASIN2.BAT

Do Not Run From Command Line

OCCASIN2.BAT is designed to be called by the AUTOEXEC.BAT file and should not be run from the command line. Every tenth time the AUTOEXEC.BAT file calls it, OCCASIN2.BAT performs a backup. Unlike OCCASION.BAT, OCCASIN2.BAT uses the mathematical abilities of Batcmd to track the number of times it is run—although this information is still stored in a 0-length file. Because it uses math rather than a series of If-tests for each possible value, OCCASIN2.BAT is better suited for longer periods. Note that the batch file counts reboots and not days.

PASSWORD.BAT

Syntax `PASSWORD [/?]`

PASSWORD.BAT is a demonstration batch file that has you enter a three-digit password by entering the ASCII values of the three digits. It is then impossible to exit PASSWORD.BAT without entering the proper password. It gives you three attempts and then locks up the computer. If you press Ctrl-Break, it locks up the computer. As configured, the password is "RON" and the password is case-sensitive.

PHONE.BAT

Syntax `PHONE` *`Nickname`* `[/?]`

PHONE.BAT uses a GOTO Nickname to jump to the section containing the phone number of interest. From there, it dials the phone for you. PHONE.BAT will abort if an invalid nickname is entered. Users will need to modify the batch file to use the phone numbers and nicknames they want to call before using it. This is the solution to problem 10 in the book.

PHONE1.BAT

Syntax PHONE1 *Nickname* [/?]

PHONE1.BAT uses a GOTO *Nickname* to jump to the section containing the phone number of interest. From there, it dials the phone for you. Unlike PHONE.BAT, PHONE1.BAT verifies that the nickname exists before issuing the GOTO *Nickname* command, so it is able to handle invalid nicknames gracefully. Users will need to modify the batch file to use the phone numbers and nicknames they want to call before using it. This is an alternative solution to Problem 10 in the book. Additionally, PHONE1.BAT must either be installed in the C:\BAT subdirectory or modified in order to work properly.

PICKDEMO.BAT

Syntax PICKDEMO [/?]

This batch file illustrates using the DOS 6 CHOICE command to get information in a batch file. PICKDEMO.BAT will only run under DOS 6 or later.

QCD.BAT

Syntax QCD *Drive Subdirectory1 ... Last Subdirectory* [/?]

QCD.BAT is useful for laptop computers, which generally have the colon and backslash keys in awkward position. QCD.BAT allows you to enter the drive and/or subdirectory path without colons or backslashes, and it still changes to the appropriate location. This is the solution to Problem 12 in the book.

RCD.BAT

Syntax RCD *Nickname* [/?]

RCD.BAT uses a GOTO %1 command to jump to the specified nickname and then changes to the appropriate subdirectory and exits. It lacks error-checking to validate that the specified subdirectory exits. Users will need to add their own subdirectories and nicknames before using the batch file.

READFILE.BAT

Syntax READFILE *ASCII-File* [/?]

Displays an ASCII file on the screen. This is a solution to Problem 6 in the book. As written, more than one file can be passed to READFILE.BAT; however, wildcards are not allowed.

REMOVE.BAT

Syntax REMOVE [*Text To Search For*] [/?]

REMOVE.BAT searches through the floppy disk catalog created by CATALOG.BAT and removes all the entries containing the text entered on the command line. It first creates a backup copy of the catalog in case there is a problem.

RETURN.BAT

Syntax RETURN [/?]

RETURN.BAT pipes a CD command followed by the current subdirectory to a file called RETURNTO.BAT. RETURNTO.BAT will then return you to the current

subdirectory when run. By running RETURN.BAT prior to transferring out of a subdirectory, you can quickly return to that subdirectory.

RETURNTO.BAT

Syntax RETURNTO

RETURNTO.BAT is the batch file created by RETURN.BAT that returns you to the subdirectory from which RETURN.BAT was last run. Command line help is not available with this batch file.

RETURN2.BAT

Syntax RETURN2 [/?]

If you store a home subdirectory in the environment under the environmental variable name HOME, issuing the RETURN2 command will return you to that home subdirectory.

RETURN3.BAT

Syntax RETURN3
RETURN3 *Path*
RETURN3 [/?]

When the RETURN3 command is followed by a subdirectory specification, RETURN3.BAT changes to that subdirectory and stores that subdirectory in the environment under the environmental variable named HOME. When the RETURN3 command is issued by itself, RETURN3.BAT changes to the subdirectory stored in the environment under the environmental variable named Home.

RONNYMD.BAT

Syntax RONNYMD *Subdirectory* [/?]

RONNYMD.BAT creates a subdirectory branching off the current subdirectory and then changes to that subdirectory.

RUNCHECK.BAT

Syntax RUNCHECK [/?]

RUNCHECK.BAT is a demonstration batch file that continually runs a diagnostic program to check out a disk drive. This is a solution to Problem 4 in the book.

SAYERROR.BAT

Syntax SAYERROR [/?]

SAYERROR.BAT displays the current errorlevel value and saves it to the environmental variable ERROR. It uses a For-loop to keep the batch file small and quick.

SENDANSI.BAT

Syntax SENDANSI *ANSI-command* [/?]

SENDANSI.BAT is a batch file that will send a single ANSI escape sequence from the command line without the user needing to enter the Escape that precedes the command. Users wanting to use SENDANSI.BAT must load ANSI.SYS in their

CONFIG.SYS file. ANSI commands containing a semicolon cannot be sent with this batch file because DOS treats the semicolon as a replaceable parameter separator and does not pass it on to the batch file.

SENDANS2.BAT

Syntax SENDANS2 *ANSI-commands* [/?]

SENDANS2.BAT is a batch file that will send up to fifty ANSI escape sequences from the command line without the user needing to enter the Escape that precedes each command. Users wanting to use SENDANS2.BAT must load ANSI.SYS in their CONFIG.SYS file. ANSI commands containing a semicolon cannot be sent with this batch file because DOS treats the semicolon as a replaceable parameter separator and does not pass it on to the batch file.

SHOWBACK.BAT

Syntax SHOWBACK [/?]

SHOWBACK.BAT list the files on the C drive that have been modified since the last backup, allowing you to decide if you need to perform a backup. This is the solution to Problem 11 in the book.

SHOWGETS.BAT

Syntax SHOWGETS [/?]

SHOWGETS.BAT is a demonstration batch file that illustrates using Batcmd commands in a batch file to get information from the user. DOBATCMD.BAT is a second demonstration batch file that illustrates using Batcmd to perform other functions.

SHOWREPL.BAT

Syntax SHOWREPL *Parameter1 Parameter2...Parameter9* [/?]

SHOWREPL.BAT is a demonstration batch file that displays the first nine replaceable parameters entered on the command line.

SHOWSHIF.BAT

Syntax SHOWSHIF *Parameter1 Parameter2...Parameter15* [/?]

SHOWSHIF.BAT is a demonstration batch file that shows the first nine replaceable parameters entered on the command line and then shows the available replaceable parameters after several SHIFT commands.

STARTDAT.BAT

Syntax STARTDAT *Name* [/?]

STARTDAT.BAT is a demonstration batch file that runs a custom database program using different configuration files depending on the name of the user.

STARTWOR.BAT

Syntax STARTWOR *Name* [/?]

STARTWOR.BAT is a demonstration batch file that runs Microsoft Word using different configuration files depending on the name of the user.

TEMPLATE.BAT

Syntax TEMPLATE

TEMPLATE.BAT is a template batch file that has the first seven lines that most batch files begin with. It contains the following:

```
@ECHO OFF
REM NAME:
REM PURPOSE:
REM VERSION:
REM DATE:
NEEDHELP %0 %1
IF ERRORLEVEL 1 GOTO HELP
```

To start a new batch file, copy TEMPLATE.BAT to the name you want to use and fill out the information on lines 2 through 5.

TESTWILD.BAT

Syntax TESTWILD *Input* [/?]

TESTWILD.BAT is a demonstration batch file that tests the first replaceable parameter to see if it contains a wildcard.

TOA.BAT

Syntax TOA *File1 File2...FileN* [/?]

For the files specified on the command line, TOA.BAT copies the .DOC and .CMP files to the A drive while deleting the associated .BAK file.

TOA-2.BAT

Syntax TOA *File1 File2...FileN* [/?]

TOA-2.BAT is a modified version of TOA.BAT that calls a subroutine called TOA-2SUB.BAT to perform the actual copying and file deleting.

TOA-2SUB.BAT

TOA-2SUB.BAT is the subroutine called by TOA-2.BAT to perform the actual copying and deleting. It is not designed to be run from the command line.

TOA-3.BAT

Syntax TOA-3 *File1 File2...FileN* [/?]

TOA-3.BAT is a modified version of TOA-2.BAT that calls itself as a subroutine rather than calling an external batch file.

UNBLANK.BAT

Syntax UNBLANK [/?]

UNBLANK.BAT sets the foreground color to bright-white and the background color to blue, undoing the screen blanking of BLANK.BAT. Note that this color assignment is permanent and will not change with a CLS command. Users wanting to use UNBLANK.BAT must load ANSI.SYS in their CONFIG.SYS file.

UNENTER.BAT

Syntax UNENTER [/?]

This batch file restores the keyboard to normal after converting the keypad for hexadecimal entry using ENTERHEX.BAT.

USEOVER.BAT

Syntax USEOVER *Command*
USEOVER

When the USEOVER command is followed by a command, that command is stored in the environment and then executed. When the USEOVER command is used alone, it reruns the command stored in the environment previously. This is the solution to Problem 8 in the book.

VOLUME.BAT

Do Not Run From Command Line

VOLUME.BAT is designed to be run by STOREVOL.BAT and should not be run from the command line. VOLUME.BAT processes the information passed to it as replaceable parameters by STOREVOL.BAT and stores the volume label in the environment under the environmental variable named VOLUME.

WINDIR.BAT

Syntax WINDIR [/?]

This batch file figures out if it is running under Windows 3.1 and, if so, stores the Windows subdirectory in the environment under the environmental variable WINDOWSDIR.

WORD2QS.BAT

Syntax WORD2QS [/?]

Automatically loads the file stored in the environmental variable COMPILE into Microsoft Word for editing.

WP.BAT

Syntax WP [/?]

Runs Microsoft Word.

\BAT subdirectories

The following subdirectories branch off of the \BAT subdirectory on the disk that comes with this book.

\BLD-ADD

The files in this subdirectory use Builder Lite to construct a utility called ADD to add one to a number passed to it on the command line.

\BLD-TIME

The files in this subdirectory use Builder Lite to construct a utility called MULTIPLY to multiply two numbers together that are passed to it via the environment.

\ENVLIB

This subdirectory contains the EnvLib Library for allowing Turbo Pascal programs to read from and write to the master copy of the environment. It was developed by Sunny Hill Software and is currently sold by TurboPower Software as part of their Turbo Professional package.

\PAS-ADD

The files in this subdirectory use Pascal to construct a utility called ADD to add one to a number passed to it on the command line.

\PAS-PLUS The files in this subdirectory use Pascal to construct a utility called PLUS to add 1 to a number passed to it on the command line. Unlike the ADD utility, the PLUS lets you specify the environmental variable to use for the results on the command line.

\BCD-TIME The files in this subdirectory use Pascal to construct a utility called MULTIPLY to multiply two numbers together that are passed to it via the environment.

\QB-ADD The files in this subdirectory use QuickBASIC to construct a utility called ADD to add 1 to a number passed to it on the command line.

\QB-TIMES The files in this subdirectory use QuickBASIC to construct a utility called MULTIPLY to multiply two numbers together that are passed to it via the environment.

Dos reference

Command	Syntax Description	Version of DOS Added	Internal or External
ANSI.SYS	A program to give extended control over the screen and keyboard.	2.0	External
APPEND	Allows programs to open the data files in the specified path as though those data files were in the current subdirectory.	3.3	External
ASSIGN	Reassigns drive letters so one drive appears to be another drive. Useful with inflexible programs that expect data to be on a specific drive. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	2.0	External
ATTRIB	Changes read-only and archive flag of files. Enhanced in DOS 3.3 to process subdirectories as well as files.	3.0	External
BACKUP	Makes copy of files on a hard disk to floppy disks. Files must be restored to be useful. Generally, a file must be restored with the same DOS version that was used to create the backup. DOS 3.0 adds the ability to automatically format disks while performing a backup and the ability to back up files modified after a certain date. DOS 3.3 adds options to back up files modified after a specified date and to create a log file. Performance was also greatly improved. DOS 4.0 adds the ability to format disks automatically without a command-line switch. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	2.0	External

Command	Syntax Description	Version of DOS Added	Internal or External
BASIC	Programming language built into ROM memory on some IBM computers. Other brands do not generally support a ROM-based Basic program. Enhanced in DOS 3.3 to be more compatible with BASICA.	1.0	Internal
BASICA	A programming language, more advanced than BASIC. Many clones will name this program BASICA or GWBASIC. Replaced in DOS 5 by QBasic. DOS 1.05 removed major bugs.	1.0	External
BREAK	Controls how often DOS checks for a Ctrl-Break from the keyboard. Can be used either as a DOS command or in the CONFIG.SYS file.	2.0	Internal
BUFFERS	Included in CONFIG.SYS file. Enhanced in DOS 3.3 so the default is tied to the machine's configuration. Enhanced in DOS 4.0 so it can read additional sectors and use expanded memory.	2.0	Internal
CALL	Runs a second batch file from within a first batch file. When the second batch file finishes, control automatically passes back to the first batch file. Any changes the second batch file makes to the environment are passed back to the first batch file (and DOS) when the second batch file terminates.	3.3	Internal
CD or CHDIR	Changes subdirectories.	2.0	Internal
CHCP	Displays or changes the active code page.	3.3	Internal
CHKDSK	A program that checks subdirectories and files against their file allocation table entries. DOS 2.0 made the corrective action (/F) optional. DOS 6 added errorlevel support where 255 indicates any error.	1.0	External
CHOICE	Accepts a keystroke from the user and reports its value via the errorlevel.	6.0	External
CLS	Clears the screen.	2.0	Internal
COMMAND	Loads a second command processor. Beginning with DOS 3.0, optionally expands the environment. (This was not documented until DOS 3.1.) In DOS 3.3, the Fail option was added to critical error messages to allow easier stopping of certain processes.	1.0	External
COMMENT	Undocumented CONFIG.SYS command that causes the CONFIG.SYS to ignore any command line beginning with the two letters specified after the COMMENT command. The syntax is "COMMENT=XY".	4.0	Internal
COMP	Compares the contents of two sets of files to see if they are the same. Beginning with DOS 2.0, it adds the use of wildcards. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	1.0	External

Command	Syntax Description	Version of DOS Added	Internal or External
COMSPEC	Included in CONFIG.SYS file. Defines an alternate path to COMMAND.COM.	2.0	Internal
COPY	Copies files from one location to another. DOS 2.0 added binary (/b) copying and verification (/v) while copying.	1.0	Internal
COUNTRY	Causes DOS to use a different country formatting for the date and currency.	3.0/3.3	Internal/ External
CTTY	Uses remote terminal in place of console.	2.0	Internal
DATE	Sets system date. External command prior to DOS 1.1 Beginning with DOS 3.3, it also sets the time in the CMOS clock.	1.0	Internal
DBLSPACE	Creates a compressed drive that DOS can access in such a way that all files are compressed before being stored to the disk.	6.0	External
DEBUG	Examines and modifies files, memory, or disk contents.	1.0	External
DEFRAG	Defragmenter that reorganizes the files on a drive to keep all the clusters of a file together.	6.0	External
DEL	Removes files from a disk. /P (prompt before erasing each file) was added in DOS 4.0.	1.1	Internal
DELOLDOS	When upgrading to DOS 6, this command removes your old version once you are sure the new version works on your system.	5.0	External
DELTREE	Deletes files and subdirectories.	6.0	External
DEVICE	Attaches a device driver. Included in the CONFIG.SYS file.	2.0	Internal
DEVICEHIGH	Loads device drivers into high memory. Requires a 80386SX, 80386, or 80486 computer and EMM386 loaded first.	5.0	Internal
DIR	Lists files on a disk or in a subdirectory. DOS 2.0 added a volume label and time to the directory display. DIR handles wildcards differently from other programs. "DIR *" will display the same files as will "DIR *.*" while other programs require the full file specification. This can cause some confusion when a batch file uses the DIR command to display files and then the DEL command to delete them. DOS 5.0 added the option to sort the display. Sorting methods are name, extension, date, and size.	1.0	Internal
DISKCOMP	Compares two floppy disks to see if they are the same.	1.0	External
DISKCOPY	Makes an exact copy of one disk on second disk.	1.0	External
DISPLAY.SYS	Device driver for displaying the character sets from other countries on the screen.	3.3	External
DO	Part of the FOR command.	2.0	Internal
DOS	DOS=HIGH loads most of DOS into high memory. The UMB option keeps certain links open, allowing memory resident software to be loaded into free high memory. This requires a 80286 or higher processor.	5.0	Internal
DOSHELP	Lists a brief summary of each DOS command.	6.0	External

Command	Syntax Description	Version of DOS Added	Internal or External
DOSKEY	Loads a small (4K) program to recall and edit prior commands and allow command aliasing.	5.0	Internal
DOSSHELL	A DOS shell and task swapping program.	5.0	External
DRIVER.SYS	Gives new names to logical devices.	3.2	External
DRIVPRAM	Modifies the parameters used for an existing drive.	3.2	Internal
ECHO	Displays a message on the screen.	2.0	Internal
EDIT	Runs the excellent ASCII editor. This editor is very much like the editor built into QBasic.	5.0	External
EDLIN	Simple ASCII file editor. Major improvements were introduced with DOS 2.0. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	1.0	External
EMM386	This is a memory manager for 80386SX, 80386, and 80486 computers. It simulates expanded memory using extended memory and it allows the loading of device drivers and memory-resident software in high memory. It also enables and disables support for the Weitek coprocessor.	5.0	External
ERASE	Removes files from a disk. Prompt before erasing each file was added in DOS 4.0.	1.0	Internal
ERRORLEVEL	Batch file command for reading the errorlevel as set by other programs.	2.0	Internal
EXE2BIN	Converts .EXE files to binary format. (Not all files can be converted.) This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	1.1	External
EXIST	Batch file command for testing if a file is present. Part of the IF statement.	2.0	Internal
EXIT	Removes the most recently loaded COMMAND.COM from memory. Has no impact if only one copy is in memory.	2.0	Internal
EXPAND	Used by DOS during installation to decompress the compressed files on the setup disk.	5.0	External
FASTOPEN	The FASTOPEN command remembers where files are located on a disk once you use them. That makes repeated access to the files much faster. The *drive* is the drive to use FASTOPEN on, it should be a hard disk. FASTOPEN can use expanded memory beginning with DOS 4.0. FASTOPEN does not adjust the location it has for files if you run a disk optimizer. Therefore, you must reboot after optimizing or DOS will not be able to find files when you have used the files before and FASTOPEN tries to help DOS locate those files.	3.3	External
FC	Compares two files and reports on any differences. It also sets the DOS errorlevel to one if the files are different.	2.0	External (MS DOS Only)
FCBS	Controls the maximum number of file control blocks that can be simultaneously open. Included in CONFIG.SYS file.	3.0	Internal

Command	Syntax Description	Version of DOS Added	Internal or External
FDISK	Initializes and partitions a hard disk. Required before formatting a hard disk. Beginning in DOS 3.3, it supports multiple DOS partitions.	2.0	External
FILES	Controls how many files the entire system can have open at once. Included in the CONFIG.SYS file.	2.0	Internal
FIND	A filter for locating lines that contain a specific set of ASCII characters. Generally used with DOS piping commands.	2.0	External
FOR	Loops through items and performs the command once per item. Generally used in batch file although it can also be used from the command line.	2.0	Internal
FORMAT	A program to prepare a diskette or hard disk for use. On a floppy disk, FORMAT performs a low-level formatting, a partitioning, and a high-level format. On a hard disk, FORMAT performs only the high-level formatting. FDISK performs the partitioning. DOS 2.0 added a volume label. DOS 3.2 added protection to prevent accidentally formatting a hard disk. You must include the drive letter to format the default drive. In addition, you must enter the volume label to format a hard disk.	1.0	External
GOTO	Batch file looping command.	2.0	Internal
GRAFTABL	Memory-resident program to load high-order characters into memory for graphic mode. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	3.0	Internal
GRAPHICS	Memory-resident program to allow printing of graphic screens. Beginning with DOS 3.3, it is compatible with thermal and LCD printers.	2.0	External
HELP	Full-screen on-line help system for DOS commands.	6.0	External
IF	Batch command to conditionally execute commands.	2.0	Internal
IN	Part of the FOR command.	2.0	Internal
INCLUDE	CONFIG.SYS file to cause one named block of CONFIG.SYS commands to process another named block of commands.	6.0	Internal
INSTALL	Runs a memory-resident program while processing the CONFIG.SYS file. Only a few memory resident programs, like SHARE, can be run this way.	4.0	Internal
INTERLNK	Performs parallel or serial transfers between two computers—these computers must be connected with a special cable. This program must run on the client system.	6.0	External
INTERSVR	Performs parallel or serial transfers between two computers—these computers must be connected with a special cable. This program must run on the server system.	6.0	External
JOIN	A command to treat all the files in one drive or subdirectory as though they were in another. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	3.1	External

Command	Syntax Description	Version of DOS Added	Internal or External
KEYB??	A memory-resident program to change the keyboard translation and date and time formats for other countries.	3.0	External
LABEL	Enters or changes a volume label.	3.0	External
LASTDRIVE	CONFIG.SYS command to specify the last drive. This command might be unnecessary in DOS 3.1 and higher. In these versions, the default last drive is E. If you specify a LASTDRIVE, you add 2K of overhead to DOS.	3.0	Internal
LH	Loads a memory resident program into high memory. This requires a 80386SX or higher processor.	5.0	Internal
LINK	Runs the LINK program. Generally used only by programmers.	1.0	External
LOADFIX	Fills the first 64K of memory so nothing loads there.	6.0	External
LOADHIGH	Loads a memory resident program into high memory. This requires a 80386SX or higher processor.	5.0	Internal
MAIL	Microsoft Mail client program. Only installed if network detected.	6.0	External
MD or MKDIR	Creates a subdirectory.	2.0	Internal
MEM	Displays the amount of memory a system has quickly. It can also display the programs and internal drivers loaded into memory.	4.0	External
MEMMAKER	Processes the CONFIG.SYS and AUTOEXEC.BAT files for 80386 or better computers and figures out how to load things to improve memory usage.	6.0	External
MENUCOLOR	Defines the menu color for a CONFIG.SYS menu.	6.0	Internal
MENUDEFAULT	In a CONFIG.SYS menu, specifies the menu item to be used as the default if the user just presses return or the menu times out plus defines the timeout period.	6.0	Internal
MENUITEM	In a CONFIG.SYS menu, defines a block of commands as an item for the menu and applies an optional title for that entry.	6.0	Internal
MICRO	Memory-resident Microsoft Mail notification program. Only installed if network detected.	6.0	External
MIRROR	Records information on disks that undelete and rebuild used to recover deleted files or formatted disks. Beginning with DOS 6, these functions built into UNDELETE.EXE. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	5.0	External
MODE	A program that controls the screen, printer, speed, display, and other important functions. Exact usage will depend on the brand of computer and the version of DOS in use. DOS 3.3 adds support for COM3 and COM4 and transfer rates up to 19.2 Kbps.	1.0	External
MORE	Program to scroll data to the screen one screen full at a time.	2.0	External
MOVE	Moves files from one location to another.	6.0	External
MSAV	DOS-based anti-virus program.	6.0	External
MSBACKUP	Full-screen backup program for DOS.	6.0	External
MSD	Diagnostics tool that displays hardware and software information.	6.0	External
MWAV	Windows-based anti-virus program.	6.0	External

Command	Syntax Description	Version of DOS Added	Internal or External
MWBACKUP	Full-screen backup program for Windows.	6.0	External
MWUNDEL	Undelete program for Windows.	6.0	External
NET	Network utility to log on and share files and printers. Only installed if network detected.	6.0	External
NLSFUNC	Used in either the CONFIG.SYS file or from the command line to support country-specific information and code-page switching.	3.3	External
NOT	Batch file command for reversing a DOS decision. If the IF test is true, then the IF NOT test is false.	2.0	Internal
NUMLOCK	Specifies the status of the NumLock key in the CONFIG.SYS file.	6.0	Internal
PATH	Controls what drives and subdirectories in which DOS searches for programs and batch files.	2.0	Internal
PAUSE	Batch file command to suspend execution until a key is pressed.	1.0	Internal
POWER	Reduces power use on computer systems that meet advanced power management requirements.	6.0	External
PRINT	Memory-resident program to print ASCII files in background while the computer is doing something else. DOS 3.0 changes the way PRINT gets CPU time for printing.	2.0	External
PRINTER.SYS	Device driver for changing the printer character set to a set for another country. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft.	3.3	External
PROMPT	Command to change the DOS prompt.	2.0	Internal
PROTMAN	Memory-resident network protocol manager. Only installed if network detected.	6.0	External
QBASIC	Loads interpretive Basic.	5.0	External
RAMDRIVE.SYS	A device driver that simulates a fast drive using memory.	6.0	External
RD or RMDIR	Command to remove an empty subdirectory.	2.0	Internal
RECOVER	Command to rescue files from a disk with bad sectors. This program is no longer included with DOS 6. However, it is available on a supplemental disk from Microsoft. WARNING: This command has the potential to do a lot of damage and should be used only by experienced users and only after making a complete backup.	2.0	External
REM	Batch file command to enter remarks in a batch file. Nothing after REM is executed.	1.0	Internal
REN or RENAME	Command to change the name of files.	1.1	Internal
REPLACE	Selectively replaces files on a target drive or subdirectory with files of the same name from a different drive or subdirectory.	3.2	External

Command	Syntax Description	Version of DOS Added	Internal or External
RESTORE	Replaces files on a hard disk with those created on floppy disks during a BACKUP. Version 3.3 of DOS adds many new features. Major enhancements made with DOS 3.0. Backups created prior to DOS 3.0 are not compatible with later RESTORE programs. Backups created prior to DOS 3.2 are not compatible with later RESTORE programs. Not compatible with DOS 6 backups.	2.0	External
SELECT	A program to pick alternate keyboard and time formats for different countries.	3.0	External
SET	A command to enter or delete variables from DOS environment. Beginning with DOS 6, can also be used in the CONFIG.SYS file.	2.0	Internal
SETVER	Causes DOS to "lie" about its version number to the specified program. That way, programs that require a specific DOS version can run under DOS 5.0. This command requires a "DEVICE=SETVER.EXE" statement in the CONFIG.SYS file.	5.0	External
SHARE	A memory-resident program that controls file sharing on a network.	3.0	External
SHELL	CONFIG.SYS command to replace COMMAND.COM with an alternate command processor or to expand the DOS environment.	2.0	Internal
SHIFT	A batch file command to make additional replaceable parameters available to DOS.	2.0	Internal
SMARTDRV	Creates a hard disk cache.	6.0	External
SMARTMON	Monitors SmartDrive performance under Windows.	6.0	External
SORT	A program that rearranges text into ASCII order. Generally used with DOS piping commands.	2.0	External
STACKS	Included in CONFIG.SYS to change the number of stack frames used to process hardware interrupts and their size. Enhanced in DOS 3.3 so the default is tied to the machine's configuration.	3.2	Internal
SUBMENU	In a CONFIG.SYS file, defines one of the main menu items as an additional menu. This is one way to get around the limitation of ten menu options.	6.0	Internal
SUBST	A command to treat a subdirectory as a drive.	3.1	Internal
SWITCHES	A CONFIG.SYS file with four functions: 1) to specify that WINA20.386 has been moved out of the root directory, 2) to cause DOS to treat an enhanced keyboard as a conventional keyboard, 3) to lock out the F5 and F8 keys during booting to kept the user from bypassing the CONFIG.SYS file and 4) to avoid the 2-second delay after DOS displays the "Starting MS-DOS" message.	5.0	Internal
SYS	Moves the hidden system files to a drive.	1.0	Internal
TIME	Sets the DOS time. External prior to DOS 1.1. Beginning with DOS 3.3 it also sets the CMOS clock.	1.0	Internal
TREE	Displays the tree structure of a drive.	2.0	External

Command	Syntax Description	Version of DOS Added	Internal or External
TRUENAME	Displays the true name of a file. For example, if you enter the command "ASSIGN A=B" and then the command "TRUENAME B:FILE" where file is actually on the A drive, the response will be "A:FILE". The command is not documented.	4.00	Internal
TYPE	TYPE *drive*:*path**file* Displays an ASCII file to the screen.	1.0	Internal
UNDELETE	Unerases the specified files. Beginning with DOS 6, also stores file tracking information for later unerasing.	5.0	External
UNFORMAT	Unformats a hard disk.	5.0	External
UNINSTAL	Restores the prior version of DOS after upgrading to DOS 6.	6.0	External
VDISK.SYS	DOS command to create an electronic drive. DOS 2.0 manual contains the source code.	3.0	External
VER	Displays the DOS version number.	2.0	Internal
VERIFY	Controls whether DOS checks data after it writes to disk.	2.0	Internal
VOL	Displays the volume label.	2.0	Internal
VSAFE	Memory-resident anti-virus program.	6.0	External
MCSETUP	Network and mail configuration program. Only installed if network detected.	6.0	External
WIN20.386	Allows Windows to run in 386 enhanced mode.	5.0	External
XCOPY	Advanced copy command. Unlike the COPY command, XCOPY sets the DOS errorlevel when it terminates and cannot be used to copy between DOS devices other than files.	3.2	External

Epilogue

I hope you have learned a lot about batch files by reading this book. If this book causes you to develop a nice batch hint or raises a batch-related question not currently answered in the book, write to me in care of TAB Books and let me know. Especially write to me if there is something in the book that is not clearly explained because my overriding goal in writing this book was to make it easy to understand. While I can't always respond to individual letters, I do my best. In addition, your hint or question just might show up in the next edition of this book. The address is

Ronny Richardson
c/o TAB Books
P.O. Box 40
Blue Ridge Summit, PA 17294

If you are active in the on-line community, you can reach me on CompuServe, where my address is *70322,3100*.

Index

& ampersand, 100
. period, 100
+ plus sign, 101
: colon, 100
< less-than symbol, 96
>> dual greater-than symbol, 96
% sign, 100 (*see also* replaceable parameters)
%% dual percent signs, 100
%variable%, 100
= single equal sign, 100
== dual equal sign, 100
¦ pipe symbol, 96

A

aborting, 47
AD command, 188, 191
ADD.EXE, 200
  Builder/Builder Lite, 292-294
  QuickBASIC, 287-290
  Turbo Pascal, 297-300
Amaral, Doug, 7, 38, 144
ANSI.SYS, 147-163, 326, 389
  color control, 149, 154-156
  cursor control, 149, 156-157
  disadvantages, 150-151
  entering commands, 150
  keyboard control, 150, 157-162
  screen control, 149
  sending commands via dot prompt, 151-154
  sending commands via environment, 151
  sending commands via typing a file, 151
  video control, 150, 157
APPEND command, 389
applications and problems, 48-49, 112-113, 182-183, 280-281
  accounting program, 49
  AUTOEXEC.BAT file, 48-49
  backup program, 280
  command program, 182
  continual diagnostic program, 112
  deleting program, 280
  file copying program, 182
  file overwriting program, 183
  file reading program, 113
  formatting program, 48
  rebooting program, 112-113
  solutions, 355-371
  telephone program, 280
archival system, 233-241
  indexing document files, 239-240
  periodic backups, 233-234
  types of backup, 234-235
ASCII, 15, 40-45
  entering high-ordered characters, 43, 45
  files, 15, 144-145
  utility file, 200, 203
ASSIGN command, 389
ATTRIB command, 259, 349, 389
AUTOEXEC.BAT, 35-36, 342-350
  %0 replaceable parameter, 343
  all-in-one approach to writing, 346-347
  arrangements, 346
  communicating with CONFIG.SYS, 337-338, 346
  date and time, 342
  memory-resident software, 346
  MODE.COM, 345
  one-per-task approach to writing, 347-349
  path=, 343-344
  prompt, 344-345
  protecting, 349-350
  verify=on/off, 345

B

backup, 4-5
  archiving, 233-241
  data files, 236-237
  deleting old files, 237-239
  incremental, 20-21, 36
  indexing document files, 239-240
  periodic, 233-234
  types of, 234-235
BACKUP command, 5, 389
  errorlevel values, 92
BASIC, 390

BASICA, 390
basic input output system (BIOS), 319
batch files
aborting, 47
advantages, 3-7
applications/problems, 48-49, 112-113, 182-183, 280-281
communicating with each other, 118
creating, 13-16, 19-21
definition, 17-18
documentation (*see* documentation)
editing, 14-15
format, 18-19
getting information from users, 132-141
messages, 34-39
naming, 8-11, 69-70
pausing, 47
pseudocode, 68-69
screen design, 143-146
source code (*see* source code)
storing, 12-13
subroutines (*see* subroutines)
utilities (*see* utilities)
Batcmd, 134-142
commands, 187-199
multiple character response questions, 136, 141
single character response questions, 134-136
BatScreen, 144-145, 200
BE command, 188, 191
BREAK command, 323, 390
BS command, 188, 191
BUFFERS command, 323-324, 329, 390
Builder, 203-205, 292-296
adding numbers, 292-294
bundling, 203
compatibility, 204
features, 204
multiplying numbers, 294-296
power/flexibility, 203-204
speed, 204
writing menus with, 203-204
Builder Lite, 205-206, 292-296
adding numbers, 292-294
multiplying numbers, 294-296

C

CALL command, 166, 172, 390
CAPITAL, 200, 203, 205
capitalization, 18, 118-122
environmental variables and, 120
program, 121
using GOTO %1, 119
CC command, 189
CD command, 218, 390
CE command, 189, 191
CH command, 189, 191
child process, 326
CHKDSK command, 4-5, 128, 390
CHOICE.COM, 132, 390
CL command, 189, 191
CLS command, 149, 150, 390
CM command, 189, 192
color, screen, 149, 154-156
command line macros, 242-246
command line recorder, 246-249
COMMAND.COM, 19, 166, 320
expanding environment with, 83
commands
ANSI.SYS, 150
Batcmd, 187-199
DOS 5.0 help feature, 27, 127
DOS reference table, 389-397
DOSKEY, 11, 311-312
echoing, 32-33
errorlevel values, 92-94
external DOS, 10-11
internal CONFIG.SYS, 9-10
internal DOS, 8-9
name reconstruction, 4
recalling with DOSKEY, 315-316
replaceable parameters as, 58-59
running DOS, 11-12
running multiple, 249-251
running occasionally, 260-264
shortening using DOSKEY, 307
simulating DOSKEY, 242-251
COMMENT command, 329-330, 390
COMP command, 390
COMSPEC, 81, 391
CONFIG.SYS
booting to different configurations, 333-338
break=on/off, 323
buffers=n, 323-324, 329
comment=, 329-330
communicating with AUTOEXEC.BAT, 337-338, 346
country=n, 324
defining command blocks, 334-335
defining menus, 335-339
device=, 324-326
devicehigh, 330, 332-333
DOS 3.3, 319-328
DOS 5.0, 329-331
DOS 6.0, 332-341
dos=, 330
drivparm=, 330
example, 339-340
FCBS=, 326
files=#, 326-327
install, 330
internal commands, 9-10
lastdrive=n, 327
numlock=, 332
rem, 331
shell=, 327
stacks=, 327-328
switches, 331
configuration
AUTOEXEC.BAT, 342-350
CONFIG.SYS, 9-10
CONFIG.SYS/DOS 3.3, 319-328
CONFIG.SYS/DOS 5.0, 329-331
CONFIG.SYS/DOS 6.0, 332-341
LEAVING.BAT, 350
WINSTART.BAK, 350
COPY command, 9, 11, 13-15, 43, 169, 391
COUNTRY command, 324, 391
CS command, 189, 192
CTTY command, 391
cursor, controlling with ANSI.SYS, 149, 156-157

D

DATE command, 391
DBLSPACE command, 391
DEBUG command, 391
DEFRAG command, 391
deleting, old files, 237-239
DELOLDOS command, 391
DELTREE command, 391
DEVICE command, 330, 391
device driver, 96, 324-326
ANSI.SYS, 147-163

DEVICEHIGH command, 332-333, 391
DIR command, 11, 31, 59, 391
directories
  maintaining C:\BAT, 252
  path (*see* PATH command)
  sub-, 218, 253-256, 387-388
disk caching, 323
DISKCOMP command, 391
  errorlevel values, 92
DISKCOPY command, 391
  errorlevel values, 93
DISPLAY.SYS, 326
DO command, 391
documentation, 25-30, 372-388
  0-9.BAT, 373
  1.BAT-6.BAT, 373
  A.BAT, 373
  ACCOUNT1.BAT, 374
  ACCOUNT2.BAT, 374
  ACCTHELP.BAT, 374
  ANSIDEMO.BAT, 374
  ANSIHIDE.BAT, 374
  ANSIKEY1.BAT, 374
  ANSIKEY2.BAT, 374-375
  AUTO-1.BAT, 375
  BLANK.BAT, 375
  BOOT-4-D.BAT, 375
  bypassing comments, 29
  CATALOG.BAT, 375
  CHECK.BAT, 375
  CHECKER1.BAT, 375
  CHECKERR.BAT, 375
  CHECKWIN.BAT, 376
  CL.BAT, 376
  COMMANDS.BAT, 376
  COMPILE.BAT, 376
  COPYBACK.BAT, 376
  COPYTHE2.BAT, 377
  COPYTHEM.BAT, 377
  COUNTENV.BAT, 377
  date, 28
  DELOLD.BAT, 377
  DISCARD.BAT, 377
  DOBATCMD.BAT, 377
  DOXCOPY.BAT, 377
  EDIT2.BAT, 378
  EDITBAT.BAT, 378
  EDITPATH.BAT, 378
  ENDLESS.BAT, 378
  ENTERHEX.BAT, 378
  FASTFIND.BAT, 378
  FINDPASS.BAT, 378-379
  FORMAT.BAT, 379
  HELP.BAT, 379
  HELP1.BAT, 379
  HELPBAT.BAT, 379
  HELPBAT2.BAT, 379
  HELPBAT3.BAT, 379
  HI-ASCII.BAT, 380
  I.BAT, 380
  INDEX.BAT, 380
  internal, 25-26
  KEYASSIGN.BAT, 380
  KEYDEL.BAT, 380
  KEYERASE.BAT, 380
  KILLZERO.BAT, 380
  LISTFILE.BAT, 380
  MACRO.BAT, 381
  MENU1.BAT, 381
  MENU2.BAT, 381
  MENU3.BAT, 381
  MOVEIT.BAT, 381
  MULTI.BAT, 381
  name, 27
  NICEPROM.BAT, 382
  NOBURN.BAT, 382
  OCCASIN2.BAT, 382
  OCCASION.BAT, 382
  PASSWORD.BAT, 382
  PHONE.BAT, 382
  PHONE1.BAT, 383
  PICKDEMO.BAT, 383
  purpose, 27
  QCD.BAT, 383
  RCD.BAT, 383
  READFILE.BAT, 383
  REMOVE.BAT, 383
  RETURN.BAT, 383-384
  RETURN2.BAT, 384
  RETURN3.BAT, 384
  RETURNTO.BAT, 384
  revision, 28
  RONNYMD.BAT, 384
  RUNCHECK.BAT, 384
  SAYERROR.BAT, 384
  self-, 27
  SENDANS2.BAT, 385
  SENDANSI.BAT, 384-385
  SHOWBACK.BAT, 385
  SHOWGETS.BAT, 385
  SHOWREPL.BAT, 385
  SHOWSHIF.BAT, 385
  STARTDAT.BAT, 385
  STARTWOR.BAT, 385
  subdirectories, 387-388
  TEMPLATE.BAT, 386
  TESTWILD.BAT, 386
  TOA-2.BAT, 386
  TOA-2SUB.BAT, 386
  TOA-3.BAT, 386
  TOA.BAT, 386
  UNBLANK.BAT, 386
  UNENTER.BAT, 386-387
  USEOVER.BAT, 387
  using labels as remarks, 28-29
  version, 27-28
  VOLUME.BAT, 387
  WINDIR.BAT, 387
  WORD2QS.BAT, 387
  WP.BAT, 387
  written, 26-27
DOS,
  5.0 command help feature, 27, 127
  6.0 choices, 132
  command reference, 389-397
  components, 321
  device drivers, 96
  DOSKEY macros, 11
  external commands, 10-11
  hierarchy of commands, 12
  modifying with DOSKEY/batch files, 306-316
  piping/redirection, 95-101
  running commands, 11-12
  subroutines prior to 3.3, 165-166
DOS command, 8-9, 330, 391
DOSHELP command, 391
DOSKEY, 392
  basics, 307
  changing functions, 308
  changing names, 308
  features, 307-308
  macros, 11
  new commands table, 311-312
  recalling command lines, 315-316
  shortening commands, 307
  simulating using batch files, 242-251
  using, 308-315
DOSSHELL, 392
DRIVER.SYS, 326, 392
DRIVPARM command, 330, 392

E

ECHO command, 32-37, 392
  characters that cannot be echoed, 38
EDIT command, 14-15, 392
editing
  batch files, 14-15
  high-ordered ASCII characters and, 45
  using ANSI.SYS, 150-151
  using Microsoft Word, 15, 45
  using WordStar, 15
EDLIN command, 14-15, 392
EMM386, 392
environment, 79-90
  determining size needed, 84
  expanding using COMMAND.COM, 83
  increasing size of, 82-84
  names, 82
  putting variables into, 85
  using in batch files, 85, 89
  using when shelling out, 86-89
  variables, 81-82, 86, 120
ERASE command, 392
error messages, 174-181
  abort/ignore/retry, 175
  abort/retry/fail, 175
  abort/retry/ignore/fail, 175
  access denied, 176
  bad command/filename, 175
  batch file missing, 176
  batch file...missing retry (Y/N)?, 176
  batch files nested too deep, 176
  cannot start command/exiting, 177
  command/filename not recognized, 175
  data error, 175
  disk full, 177
  drive not ready, 175
  environment error, 177
  environment full, 177
  file sharing conflict, 178
  filename too long, 177
  invalid path specified/check path, 178
  invalid path/path not found, 178
  label not found, 175
  miscellaneous, 178
  no free file handles/cannot start command/exiting, 177
  not enough memory, 177
  not enough memory for..., 177
  not enough memory to..., 177
  not ready error, 177
  on/off parameter required, 178
  physical media error, 175
  sector not found, 175
  syntax error, 175
  too many files open, 177
  too many open files, 177
  write protect error, 178
ERRORLEVEL command, 91-94, 122-127, 393
  definition, 105
  setting, 126-127
EX command, 189-190, 192
EXE2BIN, 392
EXIST command, 107-108, 392
EXIT command, 392
exiting, using GOTO command, 72-73
EXPAND command, 392

F

FASTOPEN command, 392
FC command, 392
FDISK command, 218, 393
file control blocks (FCBs), 326, 392
files
  archiving, 233-241
  ASCII, 15, 144-145
  deleting old, 237-239
  finding, 259-260
  indexing document, 239-240
  text, 143-144
FILES command, 326-327, 393
filters, 98
FIND command, 393
floppy disks
  catalog, 267-269
  volume label, 264-266
Flores, Steven, 253
FOR command, 74-77, 393
  local variables, 75
  quirks, 77
  wildcards in replaceable parameters, 76-77
format, batch files, 18-19
FORMAT command, 6, 11, 218, 393
formatting, safety, 6-7

G

GE command, 192
GF command, 192
GK command, 192
GL command, 192
global variables, 75
GN command, 192
GOTO command, 28-29, 70-74, 119, 175, 393
  exit feature, 72-73
  formatting option, 73
  separating code, 73
  transferring control, 72
  visually separating code, 73-74
GRAFTABL command, 393
GRAPHICS command, 393
GU command, 192

H

hard disk drives
  configuration, 219
  structuring using menus, 217-220
help, 27, 127-128
  automated system, 209-216
  context-sensitive, 214-216
  customized, 212, 214
  NEEDHELP utility, 201, 203
HELP command, 393

I

IF command, 393
IN command, 393
INCLUDE command, 393
indenting, 18
indexing, document files, 239-240
input/output (I/O), 95
INSTALL command, 330, 393
insurance, 233
INTERLNK command, 393
INTERSVR command, 393
ISITWIN, 201
ISITZERO, 201

J

JOIN command, 393

K

KEYB command, 394
  errorlevel values, 93
keyboards and keys

controlling with ANSI.SYS, 150, 157-162
DOSKEY (*see* DOSKEY)
reducing keystrokes, 3-4
simulating DOSKEY, 242-251

L

LABEL command, 394
LASTDRIVE command, 327, 394
LH command, 394
lines
blank, 37-38
length of messages, 19
naming program, 69-70
LINK command, 394
LOADFIX command, 394
LOADHIGH command, 394
local variables, 75
logic testing, 102-111
advanced, 117-130
capitalization and, 118-122
errorlevel and, 105, 122-127
file existence, 107
forms of, 102
inter-batch communications, 118
multiple IF tests, 122
NOT modifier, 117
strings, 102-105
subdirectory existence, 107-108
using help feature, 127-128
Windows and, 128-129
looping, 68-78
exiting, 72-73
FOR command, 74-77
formatting option, 73
GOTO command, 70-74
local variables, 75
separating code, 73-74
transferring control, 72
Lotus 1-2-3, 19-20

M

macros
command line, 242-246
DOSKEY, 11
MAIL command, 394
MCSETUP command, 397
MD command, 59-60, 218, 394
MEM command, 394
MEMMAKER command, 394
MENUCOLOR command, 335, 394
MENUDEFAULT command, 394
MENUITEM command, 394
menus, 217-231
defining in CONFIG.SYS, 335-339
displaying, 221
hard disk structuring, 217-220
introduction, 220
nested, 229-230
nonresident/non-nested, 221-223
resident/non-nested, 223-226
stand-alone/resident/non-nested, 226-229
types, 221-230
using Builder to write, 203-204
messages
attractive, 36-38
displaying, 34-36
length, 18
longer/fancier, 38
telling what's going on, 35
user information, 35-36
user instructions, 36
using blank lines, 37-38
MICRO command, 394
Microsoft Word, 15, 19, 45, 58
MIRROR command, 394
MODE.COM, 345, 394
MORE command, 394
MOVE command, 394
MSAV command, 394
MSBACKUP command, 394
MSD command, 394
MULTIPLY, 201
Builder/Builder Lite, 294-296
QuickBASIC, 290-292
Turbo Pascal, 300-302
MWAV command, 394
MWBACKUP command, 394, 395
MWUNDEL command, 395

N

names
documentation, 27
environment, 82
reconstructing command, 4
NEEDHELP, 201, 203
NET command, 395
NLSFUNC command, 395
NOT modifier, 117, 395
NUMLOCK command, 332, 395

P

parameters, replaceable (*see* replaceable parameters)
passwords
choosing batch file codes, 277-278
protecting, 273-277
PATH command, 34, 80-81, 120-121, 269-273, 343-344, 395
PAUSE command, 47, 143, 395
PC command, 190, 192
piping
device drivers, 96
DOS, 95-101
problems with redirection and, 99-100
symbols, 96-97, 100
using filters in, 98
POWER command, 395
power-on self-test (POST), 320
PRINT command, 395
PRINTER.SYS, 326, 395
programs
batch files (*see* batch files)
running inflexible, 257-259
source code (*see* source code)
PROMPT command, 79-80, 151-154, 344-345, 395
PROTMAN, 395
pseudocode, 68-69

Q

QBASIC, 395
QuickBASIC, 287-292
adding numbers, 287-290
multiplying numbers, 290-292

R

RAMDRIVE.SYS, 395
RB command, 190, 192
RD command, 218-219, 395
RECOVER command, 395
REM command, 25, 28, 331, 395
RENAME command, 395
REPLACE command, 395
errorlevel values, 93
replaceable parameters, 53-61, 76-77, 131
as commands, 58-59
AUTOEXEC.BAT, 343
examples, 59-60
limits, 57

replaceable parameters, *cont.*
sending to batch files, 53-56
uses, 58
using in batch files, 56-57
using more than ten, 62-67
RESTORE command, 396
errorlevel values, 94

S

safety, 6-7
screen design, 143-146
ANSI.SYS, 147-163
BatScreen, 144-145
color control, 149, 154-156
cursor control, 149, 156-157
keyboard control, 150, 157-162
saver program, 252-253
using a screen compiler, 144-145
video control, 150, 157
writing a program, 144
SD command, 190, 192
security, password protection, 273-278
SELECT command, 396
SET command, 396
SETVER command, 396
errorlevel values, 94
SH command, 190, 192
SHARE command, 396
SHELL command, 81, 326-327, 396
SHIFT command, 62, 65-66, 396
SL command, 190, 192
SM command, 191, 192
SMARTDRV command, 396
SMARTMON command, 396
SO command, 191, 193
SORT command, 396
source code
1.BAT, 222-223
A.BAT, 4, 31, 32, 59, 108
ACCTHELP.BAT, 357-358
ANSIDEMO.BAT, 148-149
ANSIHIDE.BAT, 153-154
AUTO-1.BAT, 356-357
AUTOEXEC.BAT, 35-36, 346-348
BATCH.BAT, 28-29, 176
BOOT-4-D.BAT, 359-360
CAPITAL.BLD, 205
CATALOG.BAT, 267
CHECKER.BAT, 105
CHECKWIN.BAT, 253
CL.BAT, 246-249
COMMANDS.BAT, 58-59
COPYBACK.BAT, 236-237
COPYTHE2.BAT, 363-365
COPYTHEM.BAT, 360-362
DISCARD.BAT, 238
DOBATCMD.BAT, 193-199
documentation (*see* documentation)
DOXCOPY.BAT, 106-107
EDIT2.BAT, 271-272
EDITBAT.BAT, 252
ENDLESS.BAT, 70-71
ENTERHEX.BAT, 160
FASTFIND.BAT, 260
FINDPASS.BAT, 277-278
FORMAT.BAT, 355-356
HELP.BAT, 209-210
HELP1.BAT, 210-211
HELPBAT.BAT, 213-214
HELPBAT2.BAT, 215
HELPBAT3.BAT, 216
I.BAT, 28, 36
INDEX.BAT, 240
JUNK.BAT, 290
KEYASSIGN.BAT, 309-310
KEYERASE.BAT, 312-313
KILLZERO.BAT, 370-371
LISTFILE.BAT, 235
MACRO.BAT, 243-245
MENU.BAT, 222
MENU2.BAT, 224-226
MENU3.BAT, 227-229
MOVEIT.BAT, 314-315
MULTI.BAT, 249-251
NOBURN.BAT, 253
OCCASION.BAT, 261-262
OCCASIN2.BAT, 263-264
PASSWORD.BAT, 273-276
PHONE.BAT, 365-366
PHONE1.BAT, 366-368
PICKDEMO.BAT, 133
QCD.BAT, 368-370
RCD.BAT, 256
READFILE.BAT, 360
REMOVE.BAT, 268-269
RETURN.BAT, 254
RETURN2.BAT, 254
RETURN3.BAT, 255
RONNYMD.BAT, 59, 109
RUNCHECK.BAT, 358
SAYERROR.BAT, 124-126
SETPATH.BAT, 349
SHOWADD.BAT, 290
SHOWBACK.BAT, 368
SHOWGETS.BAT, 137-141
SHOWREPL.BAT, 55-57
SHOWSHIFT.BAT, 64-65
STARTDAT.BAT, 257-258
STARTWOR.BAT, 258-259
TOA-2.BAT, 167
TOA-2SUB.BAT, 168-170
TOA-3.BAT, 171-172
TOA.BAT, 60, 66, 71, 108-110, 166-167
UNENTER.BAT, 161
USEOVER.BAT, 362-363
VOLUME.BAT, 265-266
WP.BAT, 58
spacing, 18
SS command, 191, 193
STACKS command, 327-328, 396
strings, logic testing and, 102-105
SU command, 191, 193
subdirectories, 218
changing, 253-256
documentation, 387-388
SUBMENU command, 396
subroutines, 164-173
as separate batch files, 165
definition, 165
example, 166-170
inside the calling batch file, 170-171
later than DOS 3.3, 166
prior to DOS 3.3, 165-166
storing commands within batch files, 171-172
SUBST command, 396
SWITCHES command, 331, 396
SY command, 191, 193
SYS command, 396

T

terminate-and-stay resident (TSR), 346
testing, logic, 102-111, 117-130
text files, screen design and, 143-146

TIME command, 396
TREE command, 396
troubleshooting
  booting problems, 320-321
  diagramming the logic, 179-181
  error messages, 174-181
  isolating the problem, 179
  rewriting batch files, 179
  spelling errors, 178
  splitting out problem section, 179
TRUENAME command, 397
Turbo Pascal, 296-302
  adding numbers, 297-299
  adding plus variables, 299-300
  EnvLib library unit, 296-297
  multiplying numbers, 300-302
TYPE command, 38, 397

U

UNDELETE command, 397
UNFORMAT command, 397
UNINSTAL command, 397
utilities
  ADD, 200, 287-290, 292-294, 297-300
  ASCII, 200, 203
  Batcmd (*see* Batcmd)
  BatScreen (*see* BatScreen)
  CAPITAL, 200, 203, 205
  CHOICE, 132
  contents, 285-287
  definition, 285
  ISITWIN, 201
  ISITZERO, 201
  miscellaneous, 200-201
  MULTIPLY, 201, 290-292, 294-296, 300-302
  NEEDHELP, 201, 203
  writing, 285-305

V

variables
  global, 75
  local, 75
  system-wide (*see* environment)
VDISK.SYS, 326, 397
VER command, 397
VERIFY command, 345, 397
video, controlling with ANSI.SYS, 150, 157
VO command, 191, 193
VOL command, 264-266, 397
VSAFE command, 397

W

wildcards, 76-77
WIN20.386, 397
Windows, 302-303
  logic testing for, 128-129, 253
WordPerfect, 19
WordStar, 15, 19

X

XCOPY command, 13, 20, 105, 169, 397
  errorlevel values, 94

Z

ZyIndex, 239

Other Bestsellers of Related Interest

BUILD YOUR OWN 386/386SX COMPATIBLE AND SAVE A BUNDLE—2nd Edition
—Aubrey Pilgrim

Assemble an 80386 microcomputer at home using mail-order parts that cost a lot less today than they did several years ago. Absolutely no special technical know-how is required—only a pair of pliers, a couple of screwdrivers, and this detailed, easy-to-follow guide. 248 pages, 79 illustrations. Book No. 4089, $18.95 paperback, $29.95 hardcover

BIT-MAPPED GRAPHICS—Steve Rimmer

This is one of the first books to cover the specific graphic file formats used by popular paint and desktop publishing packages. It shows you how to pack and unpack bit-map image files so you can import and export them to other applications. And, it helps you spot through available file formats, standards, patches, and revision levels, using commercial-quality C code to explore bit-mapped graphics and effectively deal with image files. 504 pages, 131 illustrations. Book No. 3558, $26.95 paperback, $38.95 hardcover

MACINTOSH SYSTEM 7: The Complete Sourcebook—Gordon M. Campbell

Campbell shows off some of the exciting new features of System 7 and offers tips for upgrading your hardware and software. This is your best guide to the first major development in the Macintosh since its introduction in 1984. With this book by your keyboard, you can count on clear skies and smooth sailing, for either upgrade or installation. 320 pages, illustrated. Includes 3.5" disk. Book No. 4074, $32.95 paperback only

THE CONCISE PC NOTEBOOK AND LAPTOP USER'S GUIDE—Dan Gookin

Here, you'll find complete information on computers designed to leave the office and follow you on the road. Useful tips are furnished throughout to help make laptop computing easier and more productive for you, no matter what your technical skill. With this book in hand, your initiation to laptop computing will be virtually painless! 304 pages, 40 illustrations. Includes 3½" disk. Book No. 3921, $22.95 paperback only

MS-DOS® BATCH FILE PROGRAMMING —3rd Edition—Ronny Richardson

Now updated to cover DOS 5.0, this book explores the power of .BAT—the PC user's key to total system control. Richardson shows how to boost productivity dramatically with simple step-saving programs. He discusses two of the most often customized system batch files, AUTOEXEC.BAT and CONFIG.SYS and then shows you how to create your own batch files. 440 pages, 186 illustrations. 5.25" disk included. Book No. 3916, $26.95 paperback, $36.95 hardcover

MS-DOS® BATCH FILE UTILITIES
—Ronny Richardson

Featuring more than 200 of the best batch file programs available for the PC, this is the most complete source of documentation available for batch file utilities currently offered as shareware or in the public domain. Arranged alphabetically and meticulously cross-referenced by category, this valuable reference features detailed descriptions and instructions for ALL commercial batch files on the DOS market today. 368 pages, 275 illustrations. 5.25" disk included. Book No. 3915, $29.95 paperback, $36.95 hardcover

FOXPRO®: The Master Reference —2nd Edition—Robin Stark and Shelley Satonin

Design and run powerful, customized databases in no time using all the exciting new features of FoxPro. This alphabetical guide to every FoxPro command and function covers all versions through 2.0—more than 350 entries in all. Its innovative three-part indexing system leads you quickly to all commands, functions, and examples found in the book. 512 pages, 135 illustrations. Book No. 4056, $24.95 paperback only

NORTON UTILITIES® 6.0: An Illustrated Tutorial—Richard Evans

Richard Evans shows you how to painlessly perform the most dazzling Norton functions using the all-new features of Norton Utilities 6.0. He also reviews the best from previous releases, providing clear, easy-to-follow instructions and screen illustrations reflecting Norton's new developments. You'll also learn about NDOS, a new configuration and shell program that replaces COMMAND.COM. 464 pages, 277 illustrations. Book No. 4132, $19.95 paperback, $29.95 hardcover

101+ FOXPRO® AND dBASE® IV USER-DEFINED FUNCTIONS—Philip Steele

Whether you've already written many lines of database code and just want to improve your code or you want to develop more complex applications for distribution in the corporate marketplace, this book's for you. It contains professional guidelines for developing and writing UDFs that will eliminate repetitive database programming tasks. A companion disk, offered on an order form at the end of the book, contains all the UDFs used in the book. 368 pages, 159 illustrations. Book No. 3951, $22.95 paperback only

COMPUTER SECURITY HANDBOOK —2nd Edition—Richard H. Baker

This edition emphasizes practical, affordable measures that protect networks and database servers, featuring all-new coverage of virus control methods, the 1986 Computer Fraud and Abuse Act, and recent case studies of security problems. You'll find complete information on prevention and cure of viruses, electronic eavesdropping, personnel controls, identifying your most vulnerable points, password perils, security planning, and how a computer can protect itself. 432 pages, 70 illustrations. Book No. 3592, $24.95 paperback, $34.95 hardcover

GRAPHICAL USER INTERFACE PROGRAMMING—Steve Rimmer

Graphical user interfaces are one of the hottest topics in PC software technology today. Creating and using GUIs requires a delicate combination of graphics programming skill and an ability to organize multiple graphic objects. This book shows you how to combine these skills to create images through bit-mapping graphics. Rimmer shows you how to write tight code menus, screen fonts, mouse interfaces, icons, and more. 440 pages, 176 illustrations. Book No. 3875, $24.95 paperback, $36.95 hardcover

THE RELATIONAL DATABASE ADVISOR: Elements of PC Database Design
—Kimberly Maughan Saunders

Kimberly Saunders gives you easy-to-follow guidelines for every phase in the database design process, providing specific suggestions for building databases that fit a variety of business needs and software environments. You'll soon create sophisticated libraries of database files with common informational links. 248 pages, 128 illustrations. Book No. 3944, $16.95 paperback only

SQL: Structured Query Language —2nd Edition
—Dr. Carolyn J. Hursch and Dr. Jack L. Hursch

Carolyn J. Hursch and Jack L. Hursch present a complete overview of SQL, tracing its mathematical structure from its basis in first-order logic to its present-day role and the efforts of the American National Standards Institute (ANSI) to develop a standard SQL language. They cover all the components of conventional SQL language; SQL commands, keywords, and data types; and value expressions supported by SQL. 216 pages, illustrated. Book No. 3803, 21.95 paperback, $32.95 hardcover

DOS 5 DEMYSTIFIED—James S. Forney

This book provides a frank appraisal of the strengths, weaknesses, and peculiarities of this new release, offering insider tips to help you get the most from DOS 5's new features. It emphasizes compatibility and productivity to help you take full advantage of the power of DOS 5. Some of the new and enhanced features covered include: HIMEM.SYS, DISK SERVICES, QBASIC, and EDIT. 440 pages, illustrated. Book No. 3860, $24.95 paperback, $34.95 hardcover

THE ENTREPRENEURIAL PC—Bernard J. David

Put that expensive home PC to work for you. You will learn about the profit-making potential of computers in typing, word processing, desktop publishing, database programming, hardware installation, electronic mail, and much more. David uses detailed, real-life examples to describe some of the more popular avenues of entrepreneurship for the home PC owner. 336 pages, 50 illustrations. Book No. 3823, $19.95 paperback, $29.95 hardcover

UPGRADE YOUR IBM® COMPATIBLE AND SAVE A BUNDLE—2nd Edition—Aubrey Pilgrim

Praise for the first edition . . .

"Every aspect is covered . . . liberally and clearly illustrated . . . invaluable." ***—PC Magazine***

Find valuable advice on adding the newest high-quality, low-cost hardware to your PC with this book. It offers informative how-to's for replacing motherboards with 80286, 80386, and 08486 boards; adding new floppy and hard disks drives; replacing old BIOS chips, installing chips and memory boards; and plugging in internal modems and VGA, fax, and network boards. 264 pages, 60 illustrations. Book No. 3828, $19.95 paperback, $29.95 hardcover

Prices Subject to Change Without Notice.

Look for These and Other TAB Books at Your Local Bookstore

To Order Call Toll Free 1-800-822-8158

(24-hour telephone service available.)

or write to TAB Books, Blue Ridge Summit, PA 17294-0840.

Title	Product No.	Quantity	Price

☐ Check or money order made payable to TAB Books

Charge my ☐ VISA ☐ MasterCard ☐ American Express

Acct. No. _____ Exp. _____

Signature: _____

Name: _____

Address: _____

City: _____

State: _____ Zip: _____

Subtotal $ _____

Postage and Handling
($3.00 in U.S., $5.00 outside U.S.) $ _____

Add applicable state and local
sales tax $ _____

TOTAL $ _____

TAB Books catalog free with purchase; otherwise send $1.00 in check or money order and receive $1.00 credit on your next purchase.

Orders outside U.S. must pay with international money in U.S. dollars drawn on a U.S. bank.

TAB Guarantee: If for any reason you are not satisfied with the book(s) you order, simply return it (them) within 15 days and receive a full refund.

BC

Order Form for Readers Requiring a Single 5.25" Disk

This Windcrest/McGraw-Hill software product is also available on a 5.25"/1.2Mb disk. If you need the software in 5.25" format, simply follow these instructions:

- Complete the order form below. Be sure to include the exact title of the Windcrest/McGraw-Hill book for which you are requesting a replacement disk.

- Make check or money order made payable to *Glossbrenner's Choice*. The cost is $5.00 ($8.00 for shipments outside the U.S.) to cover media, postage, and handling. Pennsylvania residents, please add 6% sales tax.

- Foreign orders: please send an international money order or a check drawn on a bank with a U.S. clearing branch. We cannot accept foreign checks.

- Mail order form and payment to:

  Glossbrenner's Choice
  Attn: Windcrest/McGraw-Hill Disk Replacement
  699 River Road
  Yardley, PA 19067-1965

Your disk will be shipped via First Class Mail. Please allow one to two weeks for delivery.

Windcrest/McGraw-Hill Disk Replacement

Please send me a replacement disk in 5.25"/1.2Mb format for the following Windcrest/McGraw-Hill book:

Book Title _____

Name _____

Address _____

City/State/ZIP _____

If you need help with the enclosed disk . . .

This supplementary disk contains batch files appearing in *MS-DOS Batch File Programming, 4th Ed.* (Book #4335), © 1993 by Windcrest. It should contain two files and five subdirectories:

README.BLD
README.EXE
BAT
MENU1
MENU2
MENU3
PROGRAM

Simply load your disk into your floppy drive (probably drive B), switch your prompt to that drive, and type:

```
README
```

and Enter. Then try the options on the resulting menu, for more information.

See Appendix B for specific file descriptions.

DATE DUE

SEP 2 5 1997

NOV 2 5 1998

OCT 2 8 1998

OCT 2 7 1998

GAYLORD PRINTED IN U.S.A.

JAMES B. DUKE MEMORIAL LIBRARY
JOHNSON C. SMITH UNIVERSITY
CHARLOTTE, N. C. 28216

005.43 MSR39m 1993

Richardson, Ronny.

MS DOS batch file
programming

Make DOS work for you!

Updated to cover DOS 6, ***MS-DOS Batch File Programming, 4th Edition*** gives you all the hands-on guidance you need. "Dr. Batch File" himself makes learning batch file programming simple by supplying subroutines with corresponding explanations of each command. Create all kinds of time-saving batch files including:

- A personalized menu system
- An automatic document archive system
- A library of virus prevention batch files
- TSR application managers
- Batch file utility programs
- Customized CONFIG.SYS, AUTOEXEC.BAT, WINSTART.BAT, and shutdown files

Review questions help you to gauge your progress

Features complete DOS command summaries and an ASCII character chart

Includes a disk packed with helpful batch file programs

DOS
Intermediate

Windcrest®/McGraw-Hill
Blue Ridge Summit, PA 17294-0850

ISBN 0-8306-4588-8